D.E.A. d'Informatique Fondamentale et Parallélisme

Stratégies pour un OTHELLO de haut niveau

Éric FIORIO

Septembre 1996

Laboratoire d'Optimisation Globale — École Nationale d'Aviation Civile (ENAC).

Remerciements

Je tiens avant tout à remercier Jean-Marc Alliot qui a été un excellent maître de stage durant toute cette année. Son aide me fut précieuse pour comprendre le programme OTAGE, pour discuter des améliorations possibles ou encore pour résoudre les problèmes qui survenaient. Je tiens aussi à formuler ma gratitude envers tout le Laboratoire d'Optimisation Globale, dont les divers membres furent toujours présents pour me prêter main forte ou pour quelque conseil avisé.

Mais leur secours ne fut pas que technique, et je dois bien avouer que ce fut un plaisir sur le plan humain de travailler au milieu de cette chaleureuse équipe de chercheurs.

Résumé

OTAGE est un programme de jeu d'OTHELLO. Ce programme a été développé car les jeux à deux joueurs ayant des objectifs antagonistes représentent un terrain de prédilection pour la mise en application de nombreuses techniques d'Intelligence Artificielle.

OTAGE fonctionne suivant un principe de recherche par $Iter\alpha\beta$ et utilise aussi la notion de *patterns*. OTAGE est déjà un bon programme mais nous nous proposons ici d'améliorer plus encore ses performances.

Pour ce faire, nous avons implanté diverses méthodes afin que le nombre de feuilles de l'arbre de recherche parcourues au cours d'une analyse soit moins grand : tables de réfutation, réordonnancement de l'ordre des coups. Nous avons aussi amélioré le fonctionnement des *patterns* en les rendant sensibles à la couleur du joueur qui va jouer. Enfin, nous nous sommes penchés sur les ouvertures de OTAGE qui doivent être entièrement revues.

Après implantation, nous constatons un élagage vraiment beaucoup plus important au niveau de l'arbre de recherche : OTAGE joue mieux et plus vite. Les modifications des *patterns* portent leur fruit et nous avons trouvé une autre approche plus efficace pour les ouvertures.

Table des Matières

Liste des Figures

Introduction

Nous travaillerons sur le programme OTAGE qui a été développé par Jean-Marc Alliot. OTAGE est un programme de jeu d'OTHELLO, dont les performances ne sont plus à démontrer. Il concourt d'ailleurs sur le serveur I.O.S. contre d'autres programmes, et est bien classé. Parmi les nombreux programmes présents sur ce serveur, nous pouvons noter la présence, de temps en temps, du champion du monde : LOGISTELLO.

Le but de ce D.E.A. est d'améliorer encore plus les performances de OTAGE . OTAGE joue en explorant un arbre de recherche (ou arbre des coups) au moyen d'un algorithme de type $Iter\alpha\beta$. Sa fonction d'évaluation utilise plusieurs techniques comme les degrés de liberté, une valuation des cases du plateau de jeu et aussi l'utilisation de *patterns*. Certains paramètres inhérents à ces techniques ont d'ailleurs été obtenus au moyen d'algorithmes génétiques.

Nous nous proposons d'améliorer l'$Iter\alpha\beta$ en implantant des *tables de réfutation*, en perfectionnant les *patterns* et en revoyant l'ordre dans lequel les coups sont générés. De plus, OTAGE possède diverses bases de données dont la mise à jour se fait pour l'instant de manière manuelle, par le programmeur. Nous veillerons à automatiser ces processus. Nous pourrons ensuite améliorer les ouvertures, domaine dans lequel OTAGE n'est pas très performant. Nous abandonnerons la technique du simple fichier d'ouvertures, qui présente de nombreux défauts, et utiliserons les bases de données de OTAGE afin qu'il apprenne de lui-même, "par expérience", quelles sont les ouvertures intéressantes, et qu'il développe ainsi sa propre bibliothèque. Pour cet apprentissage, nous aurons besoin de mettre en place un *autoplay*, c'est à dire un mode où OTAGE joue contre lui-même.

Dans ce rapport, nous présenterons dans un premier temps les règles du jeu d'OTHELLO et les principes de base du fonctionnement de OTAGE . Ensuite, nous effectuerons l'analyse et la conception des diverses techniques visant à améliorer OTAGE : la modification des *patterns*, la mise en place des tables de réfutation, le changement de l'ordre des coups, l'automatisation du stockage des bases de données et enfin l'amélioration des ouvertures et ses corollaires, les coups interdits et le mode autoplay. Nous détaillerons par la suite leur réalisation et montrerons les résultats obtenus.

Chapitre I

Le sujet : OTAGE

I.1 Spécifications

Nous décrivons en annexe A les règles du jeu d'OTHELLO. Le sujet de ce D.E.A. consiste à continuer à développer OTAGE, un programme de jeu d'OTHELLO réalisé par Jean-Marc ALLIOT, et d'améliorer ses performances.

Nous allons donc rechercher quelles parties du programme pourraient être améliorées, et par quelles sortes d'algorithmes nous allons réaliser ce travail. Il sera intéressant de continuer à tester OTAGE sur le serveur I.O.S. afin de poursuivre son apprentissage et d'observer son évolution en fonction des améliorations successives qui lui seront apportées.

I.2 Description de OTAGE

I.2.1 Le début de partie

En début de partie, OTAGE utilise une bibliothèque d'ouvertures qui est constituée d'un seul fichier. Ce fichier énumère tout simplement la suite des coups de chacune des ouvertures contenue dans cette base de données.

I.2.2 Le milieu de partie

Lorsque OTAGE quitte la phase des ouvertures, le programme se met à rechercher le meilleur coup à jouer à l'aide d'un algorithme de type $Iter\alpha\beta$. Cet algorithme est une variante de l'algorithme $\alpha\beta$ ($minimax$).

Nous allons expliquer ces divers algorithmes d'I.A. qui revêtent une importance primordiale pour tous les programmes de jeux à deux joueurs, à information totale et où les deux joueurs poursuivent des objectifs antagonistes.

L'arbre de recherche

Tout d'abord, précisons que dans la suite de ce rapport, nous supposerons, sauf avis contraire, que l'ordinateur joue les Noirs.

Après un coup joué par les Blancs, l'ordinateur a le choix entre plusieurs coups possibles : il s'agit de trouver le meilleur. Pour ce faire, nous allons utiliser la puissance de l'ordinateur afin de calculer plusieurs coups à l'avance et de déterminer ainsi quel coup lui assure la meilleure des situations possibles à venir.

La méthode générale est la suivante : à partir d'une position (ou état) donnée, on génère l'ensemble des positions (ou états) que l'ordinateur peut atteindre en jouant un coup (niveau 1 de profondeur). A partir de chacune de ces positions du niveau 1, on génère l'ensemble des positions que l'adversaire peut à son tour atteindre (niveau 2). On peut alors recommencer l'opération aussi longtemps que le permet la puissance de calcul de l'ordinateur et générer les niveaux 3, 4,..., *n*. On construit ainsi un *arbre de l'espace d'états du problème* ou *arbre de recherche* .

Bien évidemment, plus l'ordinateur explore de coups à l'avance, plus le nombre de positions à étudier explose. Il est clair qu'il est inimaginable de générer l'ensemble de l'espace d'états du problème complet (c'est à dire des 60 coups que comporte une partie d'OTHELLO). Une recherche exhaustive est donc impossible à mettre en œuvre. Pour preuve, nous savons qu'à OTHELLO, le *facteur de branchement*[1] est environ de 9 à chaque niveau de profondeur. Une partie d'OTHELLO contenant généralement soixante coups (64 cases moins les quatre pions de départ), auxquels on peut enlever 10 coups d'ouverture et 10 coups de fin de partie, le nombre d'états de l'espace d'états est alors de l'ordre de $9^{40} \simeq 150\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000$[2] : même l'ordinateur le plus puissant du monde ne pourrait le générer en un temps raisonnable (inférieur à l'âge de l'univers par exemple). On doit donc limiter l'exploration à une *profondeur maximale de résolution.*

Nous montrons à titre d'exemple sur la figure I-1 l'arbre généré pour la recherche du premier coup à jouer d'une partie avec une *profondeur* de 3 coups (bien qu'en fait OTAGE utilise pour les premiers coups ses bibliothèques d'ouvertures).

L'évaluation des positions

Comme nous ne pouvons nous permettre de faire une recherche exhaustive sur tous les états possibles à partir d'une position, nous disposons d'un arbre de recherche *limité à une certaine profondeur.* A nous d'en déduire le meilleur coup à jouer (dans la figure I-1 C4, D3, F5 ou E6).

Lorsqu'au cours de l'exploration de l'arbre, nous atteignons cette *profondeur*, nous ne pouvons savoir à priori si les positions étudiées (c'est à dire les feuilles de l'arbre) sont gagnantes ou perdantes. C'est pourquoi l'ordinateur attribue à chacune de ces feuilles une certaine valeur par l'intermédiaire d'une *fonction d'évaluation.*

La fonction d'évaluation La fonction d'évaluation tente d'associer à une position une valeur *estimant* la qualité de cette dernière pour l'un des deux joueurs. Plus cette valeur sera élevée, plus la position sera considérée comme "favorable" pour l'ordinateur (les Noirs[3]). Cette *estimation* peut se faire du point de vue d'un joueur fixe (convention *minimax*) ou du joueur qui a le trait à cette profondeur (convention négamax : voir plus bas L'algorithme AlphaBêta).

Les fonctions d'évaluation furent étudiées très tôt. Les premières proposées furent évidemment les plus simples. Aux dames par exemple, on peut effectuer la différence entre le nombre de ses pions et le nombre de pions de l'adversaire. Il est clair qu'une fonction heuristique élaborée permet de mieux jouer : si on inclut dans la fonction d'évaluation des notions d'attaque et de défense de pions, on améliore le niveau de jeu. En revanche, plus

[1] Le *facteur de branchement* est le nombre de nœuds moyen que l'on peut générer à partir d'une position.
[2] Ce nombre représente 150 milliards de milliards de milliards ! !
[3] Nous rappelons que tout est évalué vis à vis du camp Noir.

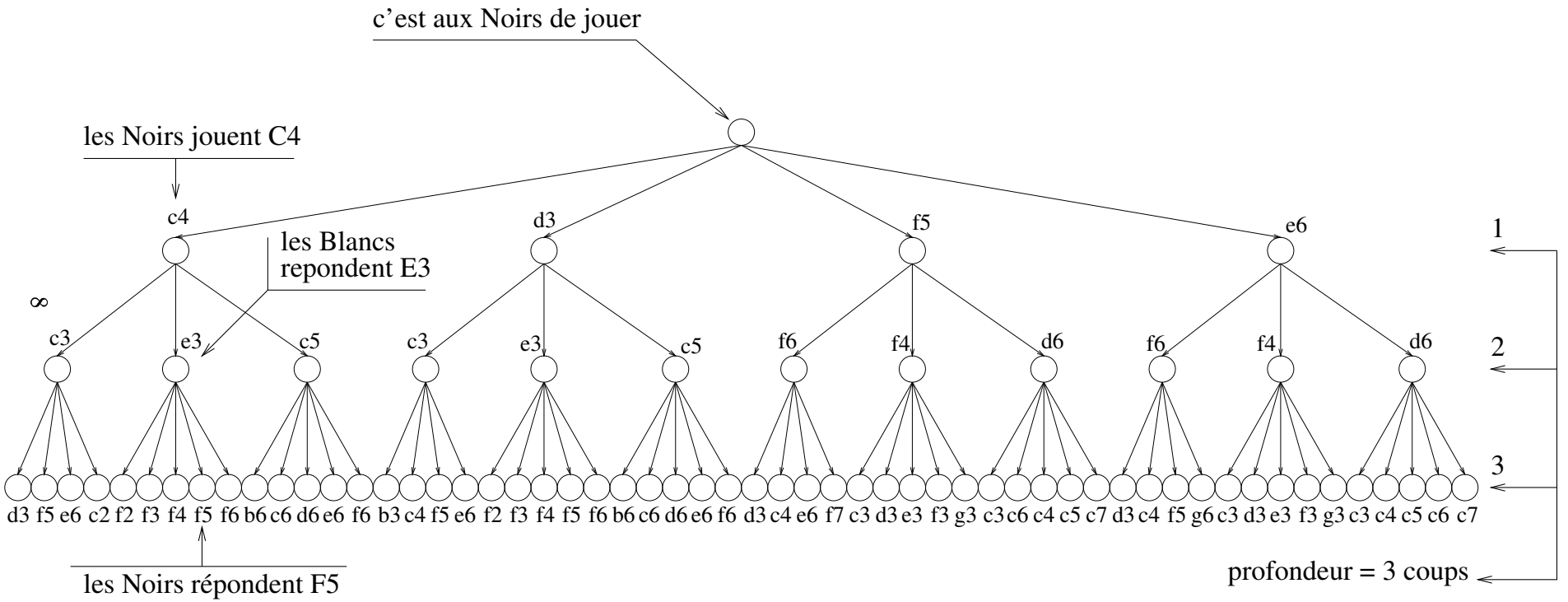


Figure I-1 : Arbre de recherche pour les trois premiers coups

la fonction est élaborée et plus il faut de temps pour la calculer, ce qui limite la profondeur d'exploration de l'arbre.

La fonction d'évaluation d'OTAGE prend en entrée la position étudiée et retourne une valeur qui dépend de nombreux critères. Elle est la somme de fonction indépendantes qui chacune est multipliée par un facteur : $f = \alpha_1 f_1 + \alpha_2 f_2 + \ldots + \alpha_n f_n$. Nous ne parlerons à titre d'information que des *degrés de liberté* des pions et des *patterns*.

Le degré de liberté Le degré de liberté d'un pion est un paramètre qui permet d'estimer la valeur stratégique du pion dans le schéma global de la position. Ce critère dépend de plusieurs facteurs liés aux pions qui l'entourent. Mais pour donner un aperçu, nous pouvons assimiler le degré de liberté au nombre de cases libres autour de ce pion (bien que ce ne soit pas exact). En effet, pour des considérations stratégiques que nous ne développerons pas ici, nous pouvons dire d'une manière générale que plus un pion est entouré, meilleure est la position de ce pion (voir figure I-2).

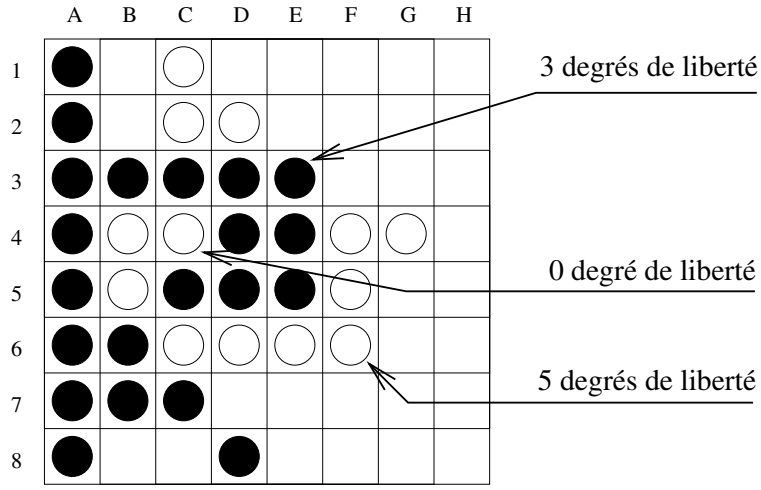


Figure I-2 : Exemple de degrés de liberté

Les *patterns* La possession des cases au bord du plateau et de celles dans les coins est primordiale et constitue une option significative pour la victoire. C'est pourquoi notre fonction d'évaluation s'intéresse tout particulièrement à ces deux types de zone du plateau. OTAGE possède une base de données où sont répertoriées toutes les configurations (ou *patterns*) possibles autour des coins (9 cases : voir figure I-3) et sur les bords du plateau (8 cases : voir figure I-3).

A chacun de ces *patterns* est associée une valeur dans la base. Plus cette valeur est élevée, plus la configuration est intéressante pour les Noirs.

Donc, au moment d'évaluer la position, OTAGE prend aussi en compte les configurations qu'il trouve aux quatre coins et aux quatre bords du plateau de jeu (voir figure I-3).

Apprentissage des paramètres Bien évidemment, il a fallu fixer des valeurs *relatives* à tous ces paramètres. Pour ce faire, on a utilisé des réseaux de neurones et des algorithmes génétiques. Ceux-ci ont généré des populations dOTHELLOS[4] jouant les uns contre les autres, et ont permis de déterminer et d'ajuster toutes ces valeurs.

[4]en fait des clones de OTAGE ayant chacun des paramètres différents

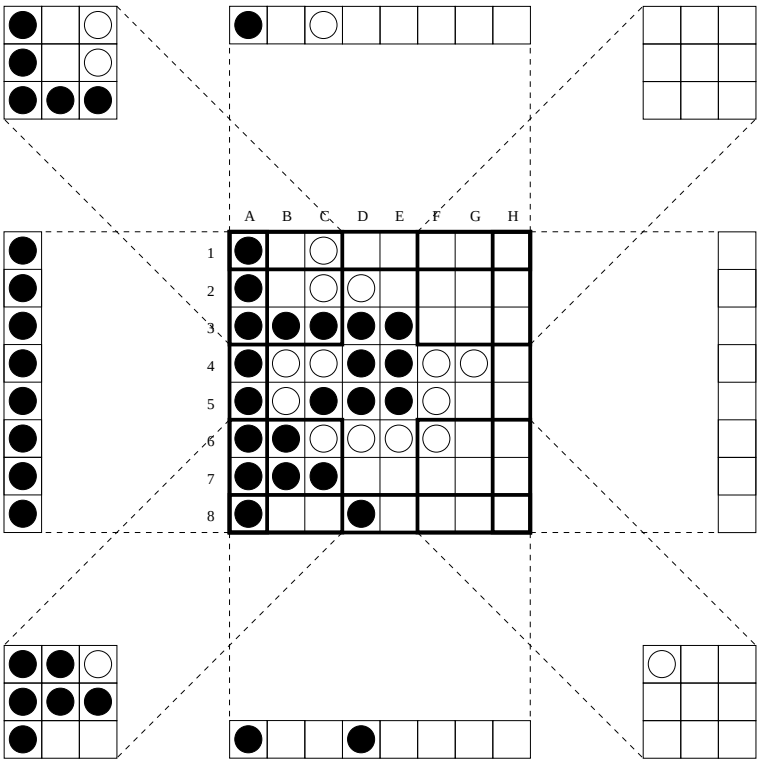


Figure I-3 : Exemple de *patterns*

L'algorithme *minimax*

A ce stade, nous avons, comme le montre la figure I-4, un arbre de recherche dont les feuilles (les positions étudiées) sont évaluées.

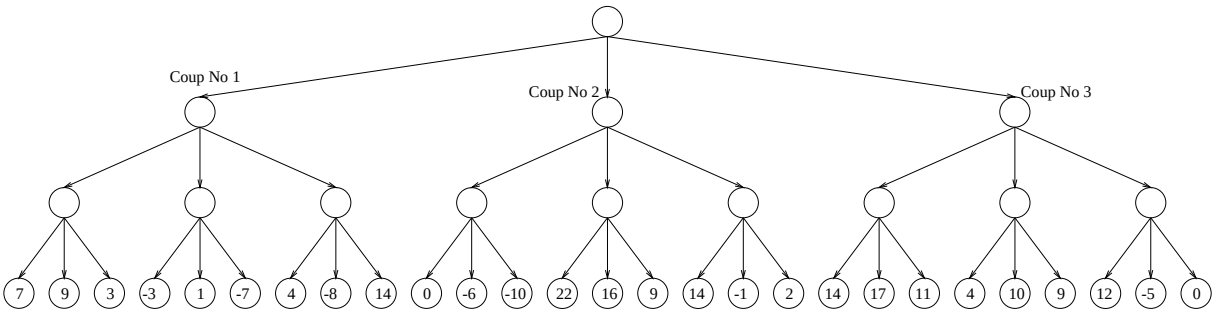


Figure I-4 : *minimax* : arbre de recherche

L'ordinateur doit alors, à partir du niveau 0 (la racine), jouer le coup de niveau 1 qui lui garantit le gain maximal *contre toute défense de son adversaire*, en supposant que celui-ci utilise également une stratégie optimale, c'est à dire qu'il joue lui-même à chaque coup le gain qui lui garantit le gain maximal contre toute défense. Ce mécanisme est appelé *principe minimax* (pour une présentation complète de cet algorithme voir [AVAD52]).

L'algorithme *minimax*[5] est un algorithme de type *exploration en profondeur d'abord*. Il

[5]Les premières descriptions de cet algorithme remontent à la fin des années 40. Ce sont Turing et Shannon qui, les premiers, ont envisagé ce mécanisme de recherche pour les ordinateurs

implante le *principe minimax* que nous avons évoqué. Le jeu comporte deux joueurs : O (l'ordinateur) et H (son adversaire). Une fonction d'évaluation h est chargée d'évaluer la qualité d'une position de jeu terminale (position de profondeur maximale ou sans descendance) du point de vue de O (par exemple). A chaque niveau où O a le trait, il peut choisir son coup et choisira donc celui de valeur maximale pour lui : on parle de nœud de type MAX. A chaque niveau où H a le trait, O va supposer que l'adversaire essaie de le mettre en mauvaise posture (minimiser ses gains) et choisira donc le coup de valeur minimale pour O (nœud de type MIN). Ce n'est que sur les feuilles que l'on peut directement appliquer la fonction d'évaluation h. Sinon, on s'appuie sur une écriture récursive. L'algorithme, qui travaille en profondeur d'abord et jusqu'à la profondeur n, ne stocke jamais plus de n positions à la fois. En effet, il commence par générer une branche complète. Il évalue alors la feuille terminale et marque le sommet n-1 avec cette valeur. Il repart alors de la position de niveau n-1 et génère la feuille suivante (niveau n). Il utilise l'évaluation de cette feuille pour modifier le marquage du niveau n-1, et répète l'opération jusqu'à ce qu'il ait généré toutes les feuilles de niveau n issues de cette position de niveau n-1. Il utilise alors la valeur de ce niveau n-1 pour marquer le niveau n-2, remonte à la position de niveau n-2 et génère la position de niveau n-1 suivante, et continue le processus jusqu'à ce que toutes les feuilles et tous les niveaux aient été générés.

Un exemple de parcours par un algorithme *minimax* de l'arbre de la figure I-4 est montré sur la figure I-5.

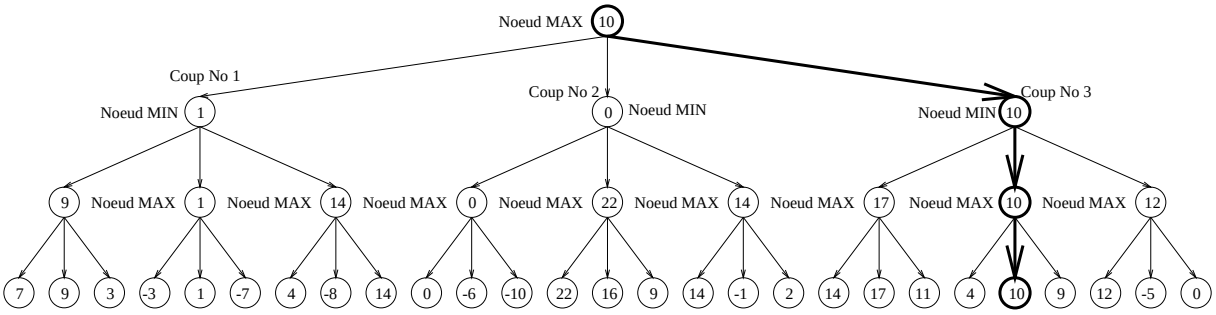


Figure I-5 : *minimax* : résultat du traitement

Pour cet exemple, nous en concluons qu'il faut jouer le coup No 3.

On constate que l'algorithme *minimax* doit complètement décrire l'arbre pour fournir une solution, ce qui, on le verra, est souvent excessif. L'algorithme $\alpha\beta$[6] est une amélioration de l'algorithme *minimax* : il réalise l'élagage de certaines branches qu'il est inutile de visiter.

Principes de l'algorithme $\alpha\beta$

L'algorithme $\alpha\beta$ est fortement inspiré du *principe minimax*, mais il réalise en plus des *coupures* de certaines branches de l'arbre : c'est à dire qu'il explore moins de branches, moins de feuilles et va donc plus vite. Comment fonctionne-t-il ?

Revenons-en au *principe minimax* appliqué à l'arbre de la figure I-4 : nous nous attarderons sur la situation de la figure I-6 qui montre ce même arbre au cours du traitement.

L'algorithme *minimax* explore exhaustivement les branches de l'arbre une à une. Sur la figure I-6, les branches déjà parcourues sont marquées en trait continu, celles non encore

[6] L'algorithme $\alpha\beta$ a été explicitement décrit pour la première fois par Hart et Edwards en 1961

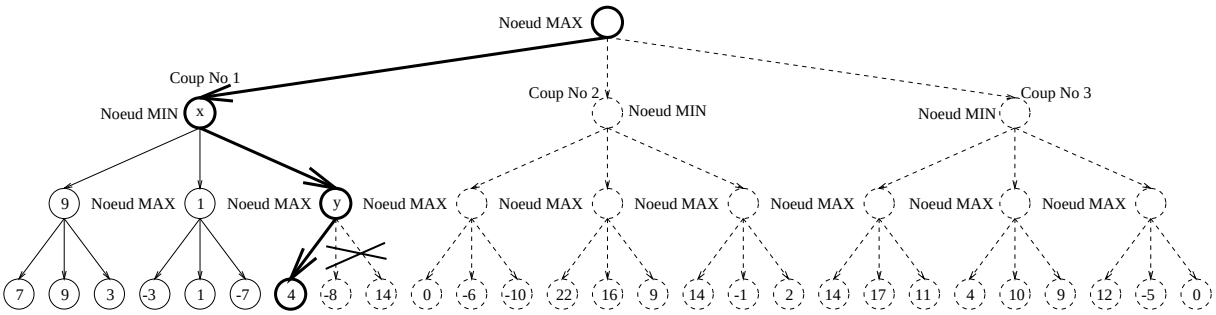


Figure I-6 : $\alpha\beta$: une coupure $\alpha\beta$

visitées en trait tireté. L'algorithme *minimax* a donc déjà analysé les feuilles de valeur respective 7, 9 et 3. Il a remonté au nœud père (nœud MAX) la valeur 9. Puis, continuant son traitement, il a fait de même avec les feuilles marquées -3, 1 et -7, pour remonter la valeur 1 vers le père. L'algorithme étudie maintenant la feuille suivante de valeur 4 (branche marquée en gras sur la figure I-6).

Si on laissait se poursuivre l'algorithme *minimax*, il passerait en revue cette feuille-là et les deux autres (-8 et 14) pour remonter vers leur père "y" (voir figure I-6) la valeur 14. Ensuite, les trois nœuds MAX étant évalués (9, 1 et y=14), l'algorithme *minimax* remonterait la valeur MIN vers leur père "x" : en l'occurrence x = 1.

Mais il n'est pas nécessaire de regarder les feuilles de valeur -8 et 14. Après avoir passé en revue la feuille de valeur 4, nous possédons suffisamment d'informations pour en conclure qu'il ne serait d'aucune utilité de poursuivre l'exploration des fils du nœud "y". En effet, le nœud "y" étant un nœud MAX, nous savons que "y" sera supérieur ou égal à 4.

$$On\ a\ : y = max(4, ..., ...) \Longrightarrow y \geq 4$$

Du coup, en ce qui concerne la valeur du nœud MIN "x", nous obtenons :

$$x = min(9, 1, y \geq 4) \Longrightarrow x = 1$$

Donc nous en déduisons directement la valeur de "x" *quelles que puissent être les valeurs des deux fils encore inexplorés du nœud "y"*. C'est ainsi que l'algorithme $\alpha\beta$ fonctionne et réalise ses coupures. On obtient finalement le résultat suivant :

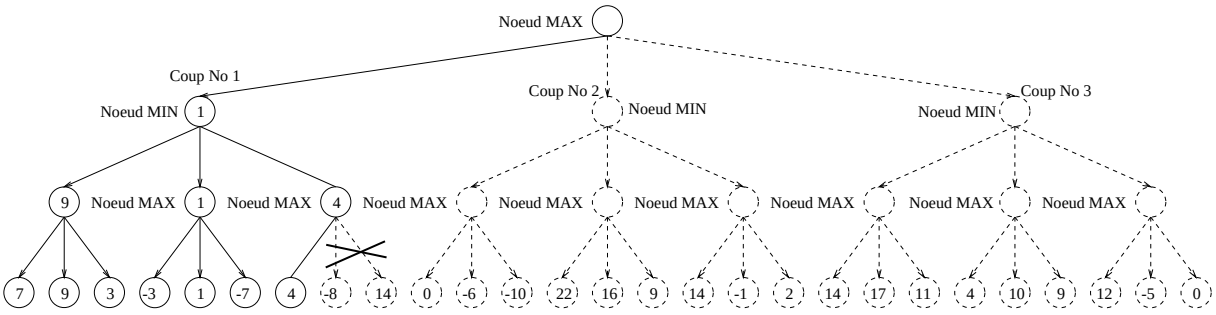


Figure I-7 : $\alpha\beta$: résultat de la première coupure

Poursuivons le traitement de la sorte, en utilisant l'$\alpha\beta$. Nous arrivons alors dans la situation de la figure I-8 où va s'opérer à nouveau une coupure, plus importante celle-là.

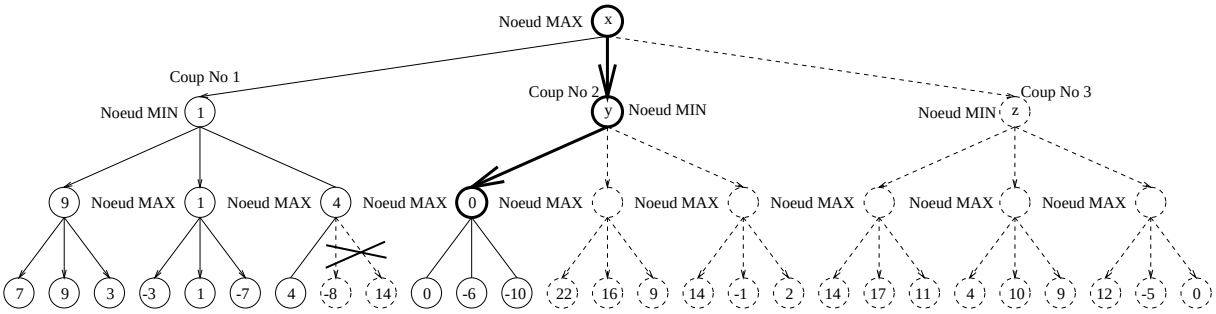


Figure I-8 : $\alpha\beta$: autre coupure $\alpha\beta$

A ce stade, nous venons juste de remonter la valeur 0 au niveau du nœud MAX (voir traits en gras sur la figure I-8). Nous pouvons faire les constatations suivantes :

$$On\ a\ : y = min(0, ..., ...) \Longrightarrow y \leq 0$$

$$et\ x = max(1, y \leq 0, z) \Longrightarrow x \geq 1$$

La poursuite de l'analyse des fils du nœud "y" ne s'impose pas ; elle n'aurait pour seul but que de déterminer la valeur définitive du nœud "y" qui de toute façon ne modifiera pas celle du nœud "x", car nous savons d'ores et déjà que $y \leq 0$! La valeur définitive du nœud "x" sera soit 1, soit celle de "z" si $z \geq 1$. On peut donc couper les branches correspondant aux autres fils du nœud "y" et l'on obtient la figure I-9.

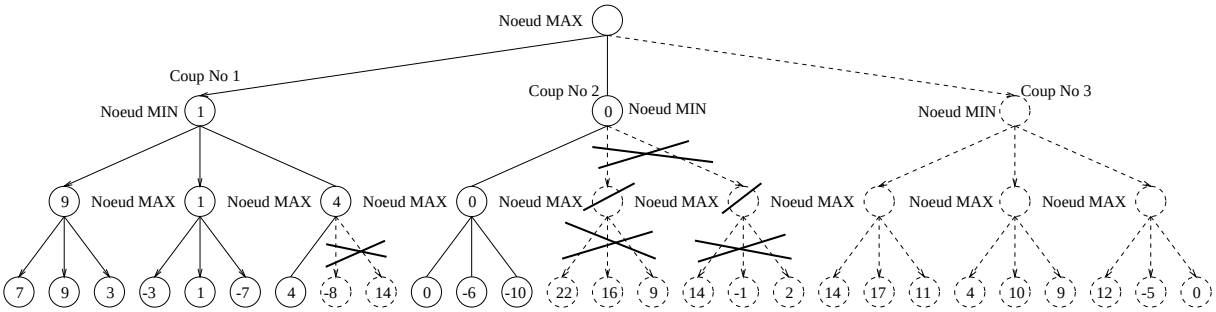


Figure I-9 : $\alpha\beta$: après la deuxième coupure

Enfin, terminons le traitement de l'arbre : le résultat final est celui de la figure I-10.

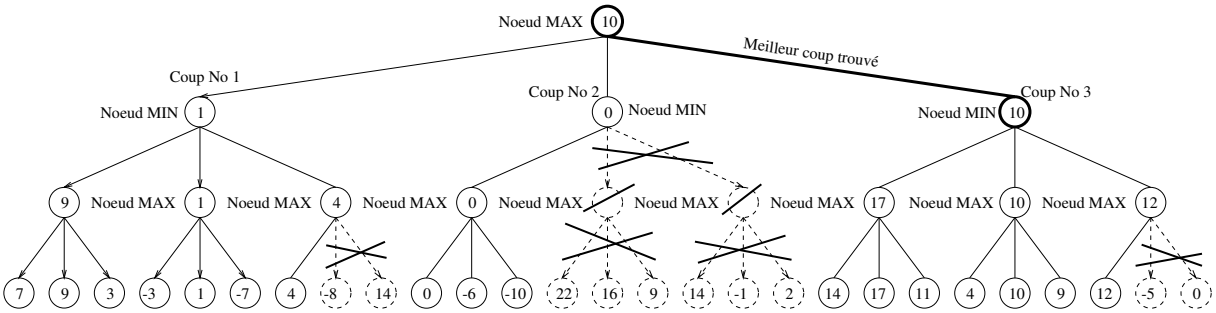


Figure I-10 : $\alpha\beta$: résultat final après passage de l'algorithme $\alpha\beta$

Le meilleur coup trouvé est bien sûr le même : le coup No 3. Mais cette fois, *nous n'avons étudié que 17 feuilles sur les 27 au total.*

L'algorithme $\alpha\beta$

De façon plus synthétique, lorsque dans le parcours de l'arbre de jeu par *minimax* il y a remise en cause de la valeur d'un nœud, si cette valeur atteint un certain seuil, il devient inutile d'explorer la descendance encore inexplorée de ce nœud.

Il y a en fait deux seuils, appelés pour des raisons historiques, α (pour les nœuds MIN) et β (pour les MAX) :

- le seuil α, pour un nœud MIN n, est égal à la plus grande valeur (connue) de tous les nœuds MAX ancêtres de n. Si n atteint une valeur inférieure à α, l'exploration de sa descendance devient inutile ;
- le seuil β, pour un nœud MAX n', est égal à la plus petite valeur (connue) de tous les nœuds MIN ancêtres de n'. Si n' atteint une valeur supérieure à β, l'exploration de sa descendance devient inutile.

Ceci nous amène à l'énoncé de l'algorithme $\alpha\beta$ qui maintient ces deux valeurs durant le parcours de l'arbre. Lors de son utilisation, on l'appellera par $\alpha\beta(Racine, -\infty, +\infty)$.

Procédure $\alpha\beta(n, \alpha, \beta)$

1. **Si** n est terminal, **alors Retourner** *h(n)*
2. **Sinon Si** n est de type MAX
   (a) **Soit** $(f_1 ... f_k)$ les fils de n
   (b) **Pour** j allant de 1 à k et **tant que** $\alpha < \beta$ **faire**
      i. $\alpha \leftarrow max(\alpha, \alpha\beta(f_j, \alpha, \beta))$
   (c) **Fin-faire**
   (d) **Retourner** α
3. **Sinon Si** n est de type MIN
   (a) **Soit** $(f_1 ... f_k)$ les fils de n
   (b) **Pour** j allant de 1 à k et **tant que** $\alpha < \beta$ **faire**
      i. $\beta \leftarrow min(\beta, \alpha\beta(f_j, \alpha, \beta))$
   (c) **Fin-faire**
   (d) **Retourner** β
4. **Fin-si**

Une définition beaucoup plus concise, mais peut-être moins intuitive, peut être obtenue en utilisant la convention négamax ; il devient alors inutile de distinguer les nœuds MIN des nœuds MAX. Avec les mêmes notations que précédemment, l'algorithme en "négamax" s'écrit :

Procédure $\alpha\beta(n, \alpha, \beta)$

1. **Si** n est terminal, **alors Retourner** *h(n)*
2. **Sinon**

(a) **Soit** $(f_1 ... f_k)$ les fils de n

(b) **Pour** j allant de 1 à k et **tant que** $\alpha < \beta$ **faire**

i. $\alpha \leftarrow max(\alpha, -\alpha\beta(f_j, -\beta, -\alpha))$

(c) **Fin-faire**

(d) **Retourner** α

3. **Fin-si**

Cet algorithme calcule alors la valeur négamax de la racine qui est, au signe près, égale à sa valeur *minimax*.

Approfondissement itératif

Le programme OTAGE utilise l'algorithme $\alpha\beta$ décrit ci-dessus mais sous une forme différente appelée *approfondissement itératif* ou encore $Iter\alpha\beta$.

L'approfondissement itératif est introduit par Scott[Sco69] afin de contrôler le temps pris par la recherche : plutôt que de commencer la recherche à la profondeur N, en ne sachant pas combien de temps elle va prendre, on effectue une succession de recherches aux profondeurs 1, 2, 3, etc..., jusqu'à ce que le temps alloué soit atteint. Le meilleur coup à la profondeur i étant évalué en premier à la profondeur $i+1$.

Au fur et à mesure que les années passèrent, cette heuristique fut raffinée et améliorée jusqu'au point où l'approfondissement itératif jusqu'à la profondeur N devint plus rapide que la recherche directe au niveau N[SA77].

La gestion du temps avec l'approfondissement itératif

Comme nous l'avons précisé, l'approfondissement itératif nous permet de gérer le temps imparti à la machine pour jouer son coup. Le principe en est le suivant : avant de se lancer dans la recherche du meilleur coup à jouer, le programme s'octroie un temps maximum pour y réfléchir. Nous ne détaillerons pas comment il fixe cette durée, mais elle dépend bien sûr du temps global lui restant pour toute la partie, du nombre de coups déjà joués, etc. Ensuite on lance l'approfondissement itératif. Une fois l'examen de la profondeur d effectué, il faut savoir si l'on aura assez de temps pour effectuer la recherche à la profondeur $d+1$. Pour ce faire, précisons que si l'arbre est parcouru exactement dans le bon ordre, le nombre de feuilles examinées est environ $2\sqrt{N}$ où N est le nombre de feuilles total de l'arbre (alors que l'algorithme *minimax* examine, lui, les N feuilles). De plus, nous savons que le *facteur de branchement* est de l'ordre de 9. Donc, d'après le nombre de feuilles explorées à la profondeur d et le temps mis pour faire cette exploration, nous sommes en mesure de produire une *bonne estimation* du temps requis pour exécuter l'$\alpha\beta$ à la profondeur $d+1$. Bien sûr, si ce temps est inférieur au temps qu'il nous reste pour réfléchir à ce coup, on stoppe là l'analyse et on utilise les résultats fournis par la recherche à la profondeur d.

Fenêtres aspirantes

Une des améliorations de l'approfondissement itératif que possède OTAGE est de faire une supposition sur la valeur *négamax* de la racine avant une recherche avec une profondeur d en fonction des valeurs *négamax* obtenues pour les profondeurs 1, 2, ..., d-1. Cette approximation prend la forme d'un intervalle $[\alpha_0, \beta_0]$ dans lequel est supposée se trouver la valeur de la racine. L'heuristique consiste alors à limiter l'intervalle de recherche de

l'algorithme *négamax* à cet intervalle plutôt que de prendre l'intervalle complet $[-\infty, +\infty]$. Trois cas peuvent se produire selon la valeur v de la recherche avec l'intervalle $[\alpha_0, \beta_0]$:

1. $\alpha_0 < v < \beta_0$ alors, l'approximation est correcte, et la recherche prend moins de temps qu'avec la fenêtre $[-\infty, +\infty]$;
2. $v \leq \alpha_0$, alors pour connaître la valeur *négamax* de la racine, il faut effectuer une recherche à la profondeur d avec une fenêtre $[-\infty, v]$;
3. $v \geq \beta_0$, alors pour connaître la valeur *négamax* de la racine, il faut effectuer une recherche à la profondeur d avec une fenêtre $[v, +\infty]$.

Bien que parfois il faille donc rappeler l'$\alpha\beta$ à la même profondeur avec une autre fenêtre, cette stratégie s'avère efficace et OTAGE gagne par ce biais beaucoup de temps sur une version sans fenêtre aspirante. Bien évidemment, l'efficacité de la technique des fenêtres aspirantes dépend de la qualité de l'approximation.

I.2.3 La fin de partie

On définit une *limite* à partir de laquelle on se considère en fin de partie. Cette *limite* dépend de plusieurs facteurs mais peut être assimilée au nombre de cases libres restant sur le plateau de jeu, c'est à dire à peu de choses près le nombre de coups restant à jouer.

Lorsque OTAGE atteint cette *limite*, il passe alors en mode "fin de partie". Dans ce mode, OTAGE effectue une recherche exhaustive sur tous les coups possibles, jouant toutes les suites de coups jusqu'à la fin de la partie. Il détermine ainsi s'il a perdu la partie (tous les coups jouables sont perdants), s'il y a match nul, ou bien si parmi la liste des coups jouables se trouvent un ou plusieurs coups gagnants.

Remarque Nous avions précisé au début de ce rapport (I.2.2 / L'arbre de recherche et I.2.2 / L'évaluation des positions) qu'il était impossible d'effectuer une recherche exhaustive vu l'explosion combinatoire du nombre de suites de coups possibles. Paradoxalement, il est possible d'effectuer une telle recherche en fin de partie bien que l'ordre de grandeur des profondeurs explorées va de 12 à 17 coups. Les raisons en sont les suivantes :

- bien que les profondeurs soient grandes, le nombre de coups jouables est faible car il ne reste plus beaucoup de cases jouables ;
- il y a souvent des "passe" en fin de partie ;
- la fonction d'évaluation change et est plus simple à calculer (voir ci-après).

En mode "fin de partie", OTAGE effectue ses recherche toujours par $Iter\alpha\beta$ mais avec cette fois une profondeur non limitative. L'$\alpha\beta$ va donc analyser toutes les suites de coups jusqu'à la fin de la partie. Par conséquent, nous ne sommes plus intéressés par la valeur *estimée* des positions qui est retournée par la *fonction d'évaluation*. En mode "fin de partie", la *fonction d'évaluation* change et retourne cette fois la différence de pions entre ceux de l'ordinateur et ceux de son adversaire. Ceci est logique car en fin de partie, nous cherchons juste à trouver si le coup analysé va nous permettre de gagner. La valeur retournée est donc entière et comprise dans l'intervalle $[-64, +64]$.

Voici comment OTAGE procède :

1. Dans un premier temps, l'$\alpha\beta$ est appelé avec une fenêtre $[-1, +1]$. Cela permet d'élaguer *énormément* de branches de l'arbre et l'on obtient *très vite* l'évaluation v de chaque coup :

   (a) si $v = -1$, le coup est perdant et l'on poursuit cette étape No 1 à la recherche d'un coup gagnant ; si on a déjà passé en revue tous les coups, alors on passe à l'étape No 2 ;

   (b) si $v = 0$, le coup mène à un match nul et l'on poursuit l'étape 1 ; si on a déjà passé en revue tous les coups, alors on passe à l'étape No 2 ;

   (c) si $v = +1$, le coup est gagnant et l'on passe directement à l'étape No 2 suivante ;

2. Ensuite, s'il reste encore du temps, OTAGE va repasser une seconde fois sur la liste des coups jouables, faisant encore une recherche exhaustive mais cette fois avec une fenêtre d'intervalle $[+1; +64]$ s'il sait qu'il a gagné, et $[-64; -1]$ s'il se sait perdant. Le but de cette deuxième recherche est le suivant :

   - si la partie est gagnée, c'est à dire s'il y a au moins un coup gagnant, OTAGE va choisir grâce à cette deuxième recherche le coup lui permettant de gagner avec le plus de pions d'avance sur son adversaire ;
   - si la partie est perdue, c'est à dire s'il n'y a pas de coup gagnant, OTAGE va choisir par ce biais le coup lui permettant de perdre avec le moins possible de pions d'écart ;

Nous n'effectuons pas cette recherche avec une fenêtre d'intervalle $[+1; +64]$ ou $[-64; -1]$ en premier, car celle-ci prend beaucoup de temps puisqu'elle n'élague pas violemment. En effet, cette fenêtre est étendue à toute la plage des valeurs. Les coupures se font donc moins nombreuses.

Le but de l'étape No 1 est donc de déterminer très rapidement si la partie est gagnée ou perdue. L'étape No 2 contente juste d'affiner l'analyse en choisissant soit le meilleur coup gagnant, soit le moins mauvais des coups (pour cette deuxième alternative, ils sont forcément tous perdants)

Chapitre II

Analyse et conception

Après avoir bien analysé et compris le fonctionnement de OTAGE, nous nous interrogeons dans ce chapitre sur les améliorations que nous pouvons apporter à ce logiciel.

II.1 Une base de données supplémentaire

Au stade où nous commençons cette analyse, OTAGE a déjà joué un peu plus de 300 parties. Il serait donc intéressant de réaliser une base de données de ces parties. Cette base de données prendrait la forme d'un arbre des coups joués au cours des diverses parties (voir figure II-1).

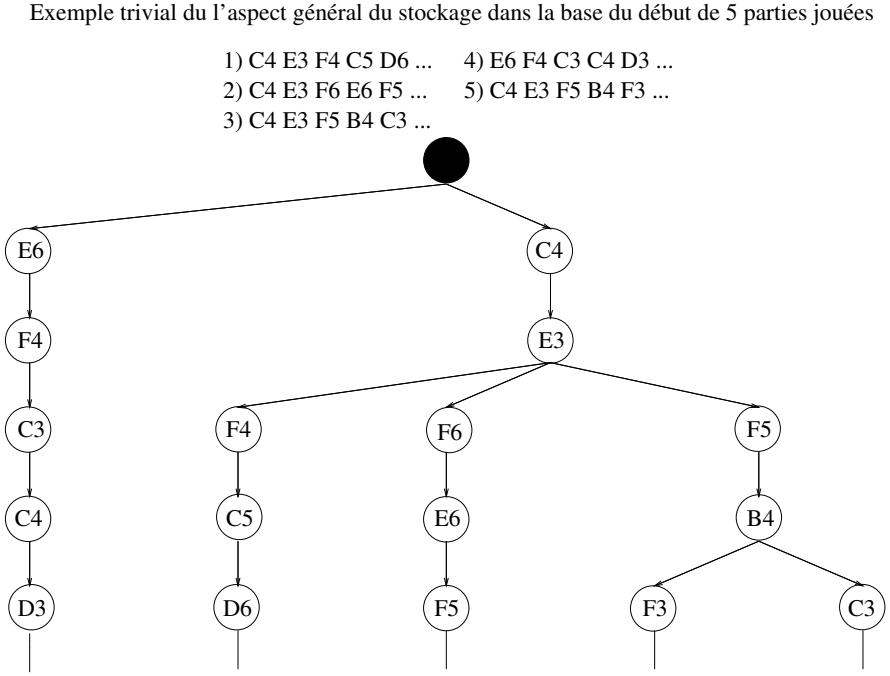


Figure II-1 : Aspect de la base de données supplémentaire

Nous aurions alors une image globale de la façon dont OTAGE joue. Nous saurions quelles ouvertures il utilise, le pourcentage de parties gagnées ou perdues en fonction de chaque ouverture. Enfin, nous pourrions apprécier le taux de réussite global de OTAGE.

Cet historique, stocké sous la forme d'une base de données, devra pouvoir être consulté par un humain.

II.2 Modification des *patterns*

Les *patterns* dont nous avons parlé au I.2.2 / L'évaluation des positions / Les *patterns*, ont été implantés indépendamment de la couleur de celui qui a le trait. Or, cette information est vitale. Certains *patterns* sont très intéressants si c'est aux Noirs de jouer, et très défavorables si c'est aux Blancs de jouer. Dans l'état actuel des choses, la valeur d'un tel *pattern* est moyennée dans la base de données. Car, cette valeur qui lui est attribuée résulte des hautes valeurs qui lui ont été assignées lorsque les Noirs ont rencontré cette *pattern* et des faibles valeurs affectées par les Blancs quand ils ont rencontré celui-ci.

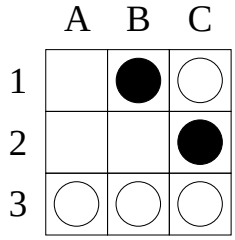


Figure II-2 : Exemple de *pattern* dépendant de la couleur

Par exemple, sur la figure II-2, il est clair que cette position est mauvaise pour les Noirs. Les Blancs menacent fortement de prendre le coin A1. Cependant, la valeur attribuée à ce *pattern* doit être différente suivant la couleur de celui qui va jouer.

Si lorsqu'on estime la position, cela va être aux Blancs de jouer, ils vont prendre la case A1. Si ce sont les Noirs qui vont jouer, la position peut être sauvée si on jouant son coup, Noir arrive à prendre le pion en C1.

Donc, la valeur attribuée à ce *pattern* doit être plus faible (c'est à dire plus catastrophique pour les Noirs) si ce sont les Blancs qui vont jouer, que la valeur si c'est aux Noirs de jouer.

Il va donc falloir implanter une base de données pour les *patterns* qui prenne en compte toutes ces remarques, et modifier en fonction l'algorithme de calcul d'estimation des positions.

II.3 Optimisation de l'$Iter\alpha\beta$

Le but de cette partie est de trouver quelles nouvelles techniques pourraient être mises en œuvre afin de diminuer encore plus le nombre de feuilles parcourues au cours de la recherche du meilleur coup à jouer. Ceci est primordial pour deux raisons principales :

- la majeure portion d'une partie d'OTHELLO est concernée par cette recherche par $\alpha\beta$.
- Si l'on explore moins de feuilles pour un même résultat, on dépense donc moins de temps. Or c'est bien ce critère qui est limitatif. Gagner du temps permettra à OTAGE d'utiliser ce surplus pour lancer une recherche à une profondeur plus grande (cf. I.2.2 / Approfondissement itératif) ou de le mettre à profit pour la recherche exhaustive de fin de partie (cf. I.2.3).

Rappel

Actuellement, l'*approfondissement itératif* se déroule comme suit : nous explorons l'*arbre de recherche* à la profondeur d. Puis, nous rappelons l'algorithme $\alpha\beta$ avec une profondeur d+1. Mais nous utilisons le résultat de l'analyse à la profondeur d pour examiner en premier, à la profondeur d+1, le meilleur coup qui avait été trouvé. Comme il y a de grandes chances pour que ce soit aussi le meilleur coup pour la profondeur d+1, nous élaguerons ainsi beaucoup de feuilles. Et même si ce n'est pas le meilleur coup pour la profondeur d+1, ce coup étant tout de même de bonne qualité, nous élaguerons de toute façon beaucoup plus de feuilles que si nous examinions en premier n'importe quel coup au hasard.

Principe

L'idée est de ne pas se contenter juste du meilleur coup ramené par l'analyse à la profondeur précédente, mais d'essayer de récupérer beaucoup plus d'informations sur l'*arbre de recherche* afin d'explorer encore plus intelligemment celui-ci à la profondeur suivante.

Dilemme

Procéder ainsi peut être risqué. En effet, si l'on décide de stocker plus d'informations au cours d'une recherche à une profondeur d afin de les utiliser pour la profondeur d+1, cela suppose un traitement supplémentaire au cours de l'exploration de l'arbre par l'$\alpha\beta$. Il faut donc s'assurer que le gain de temps obtenu à la profondeur d+1 dû à l'utilisation des renseignements fournis par la recherche à la profondeur d ne soit pas effacé par la perte de temps due à cette collecte d'informations à la profondeur d.

II.3.1 Récupération de la *meilleure branche*

Consécutivement à une recherche à une certaine profondeur d, nous pensons ramener non seulement le meilleur coup, mais aussi la *meilleure branche*. Nous entendons par cette locution, la suite (forcément issue du meilleur coup) de tous les coups de l'arbre de recherche ayant abouti à la feuille qui a fourni cette meilleure valuation au meilleur coup. La *meilleure branche* est donc la liste du meilleur coup à jouer par l'ordinateur, suivi de la meilleure réponse estimée de son adversaire, suivi de la meilleure réponse estimée de l'ordinateur à ce deuxième coup, suivi de la meilleure réponse à ce troisième coup, etc..., jusqu'au coup menant à la position terminale qui a été choisie par l'algorithme de recherche. Ainsi, lors de la recherche suivante à la profondeur d+1, nous pourrons examiner d'abord toute cette suite de coups, au lieu de n'examiner que le meilleur coup seulement.

Montrons au travers d'un exemple l'intérêt d'un tel procédé. Supposons qu'à partir d'une certaine position, nous faisions une première recherche de profondeur de 2. Avant traitement, nous avons l'arbre de la figure II-3. Appliquons à cet arbre l'algorithme $\alpha\beta$ et

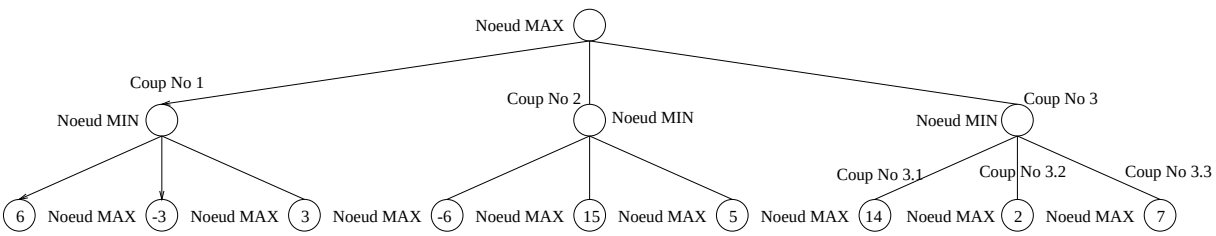


Figure II-3 : Récupération de la *meilleure branche* : l'arbre en profondeur 2

nous obtenons le résultat de la figure II-4.

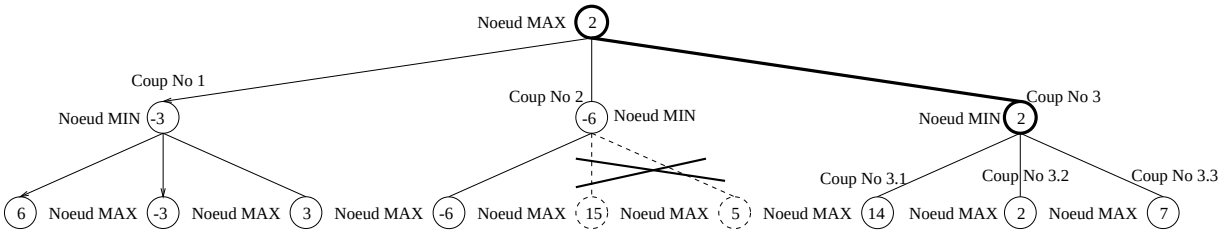


Figure II-4 : Récupération de la *meilleure branche* : l'arbre après passage de l'$\alpha\beta$

C'est donc le coup No 3 qui a été choisi. Remarquons que la *meilleure branche* est **(coup No 3, coup No 3.2)**. Maintenant, si l'on applique le principe de l'approfondissement itératif (et s'il reste assez de temps), nous allons rappeler l'algorithme $\alpha\beta$ à la profondeur 3 cette fois. Avant traitement, l'arbre a l'aspect montré sur la figure II-5.

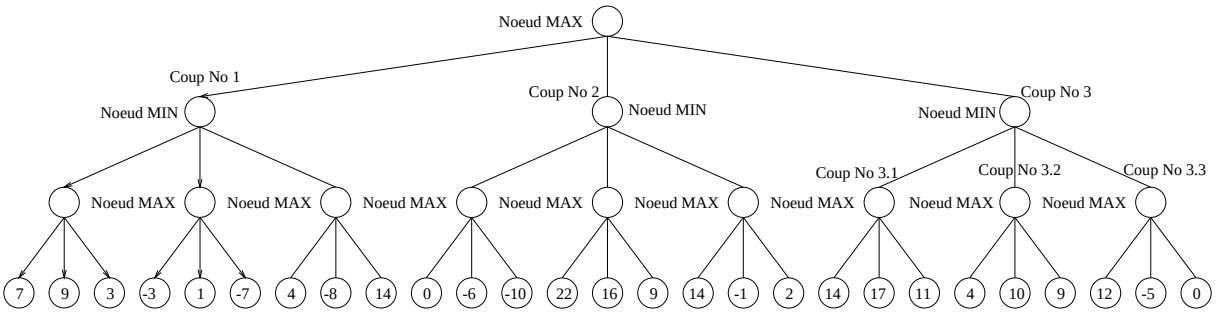


Figure II-5 : Récupération de la *meilleure branche* : l'arbre de profondeur 3

Nous allons comparer le traitement utilisant juste le meilleur coup, et celui utilisant la *meilleure branche*.

Meilleur coup

Le traitement de l'arbre va commencer cette fois non pas par le coup No 1 mais bien par le coup No 3 qui est le meilleur coup ramené par l'analyse de profondeur 2. En revanche, pour le choix du premier fils du coup No 3 à traiter, on procède comme d'habitude en étudiant les fils d'un nœud de gauche à droite. La première branche examinée sera donc **(coup No 3, coup No 3.1, feuille de valeur 14)**, comme le montre la figure II-6.

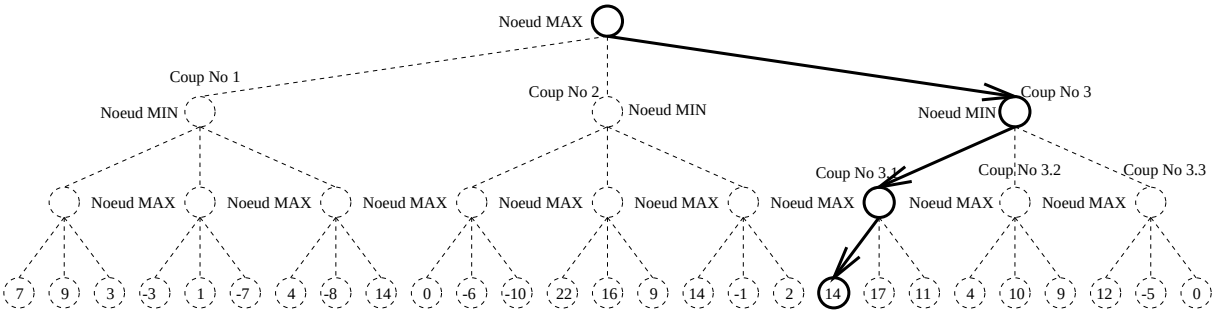


Figure II-6 : *Meilleur coup* : première feuille étudiée

Ensuite, une fois déterminée la première feuille étudiée, l'algorithme $\alpha\beta$ poursuit son traitement habituel. Nous lui avons juste forcé le premier coup à analyser. L'$\alpha\beta$ va donc traiter toutes les feuilles issues du coup No 3 pour ensuite s'attaquer au coup No 1 (voir figure II-7), pour terminer par le coup No 2.

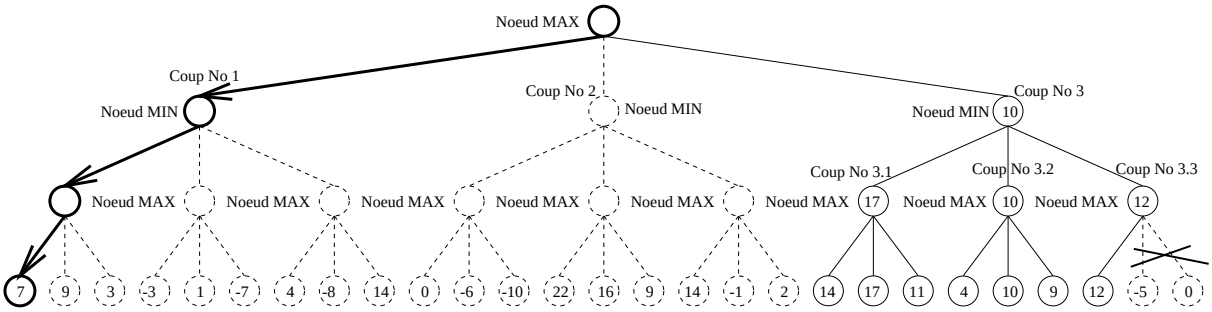


Figure II-7 : *Meilleur coup* : en cours de traitement

Le résultat final est montré sur la figure II-8.

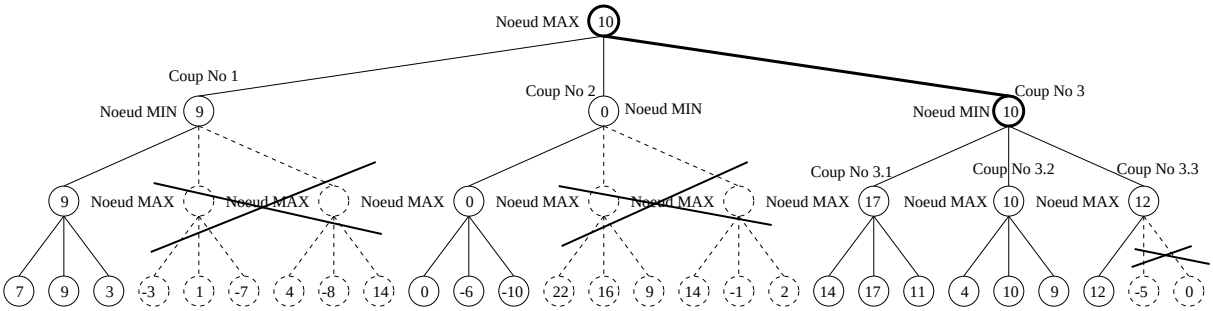


Figure II-8 : *Meilleur coup* : après traitement

Plusieurs remarques sont à faire :

1. le meilleur coup trouvé reste le coup No 3 ;
2. un algorithme de type *minimax* aurait parcouru les 27 feuilles ;
3. un algorithme de type $\alpha\beta$ sans *meilleur coup* aurait parcouru 17 feuilles sur les 27 ;
4. l'algorithme de type $\alpha\beta$ avec *meilleur coup* parcourt 4 feuilles de moins que la version précédente ; soit 13 feuilles sur 27.

Meilleure branche

Cette fois, nous allons commencer le traitement de l'arbre non pas en imposant seulement le premier coup (le coup No 3) mais toute la première branche. Nous rappelons que la *meilleure branche* ramenée par l'étude de profondeur 2 est **(coup No 3, coup No 3.2)**. La première branche étudiée sera donc celle constituée du coup No 3, suivi du coup No 3.2, suivi de la première feuille du coup 3.2 (voir figure II-9).

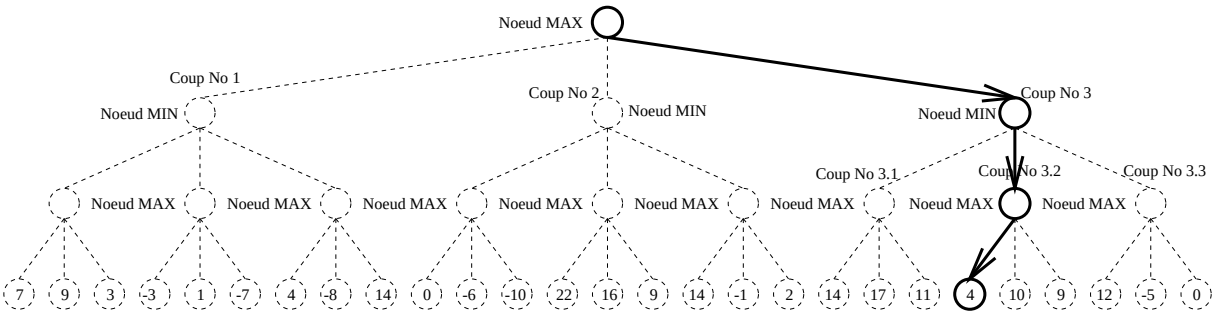


Figure II-9 : *Meilleure branche* : première feuille étudiée

L'algorithme $\alpha\beta$ opère son traitement et passe en revue les 3 feuilles du coup No 3.2. Il consulte ensuite un autre fils du coup No 3, c'est à dire le coup No 3.1. On obtient la figure II-10.

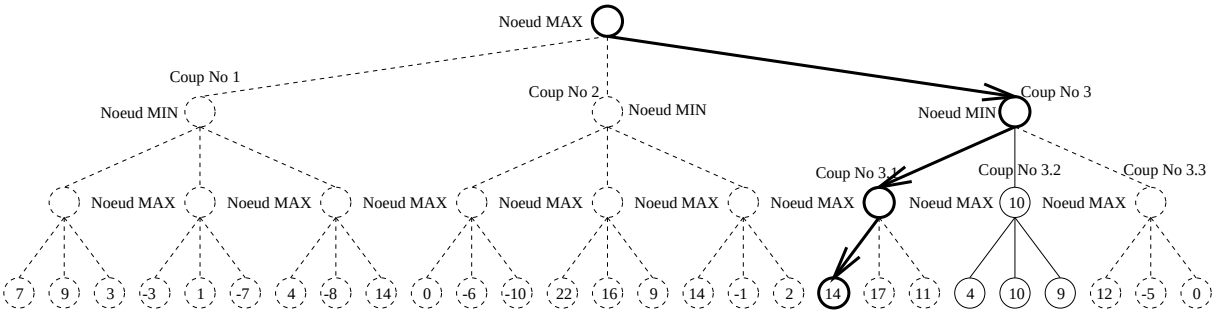


Figure II-10 : *Meilleure branche* : en cours de traitement

L'$\alpha\beta$ visite ensuite les feuilles du coup No 3.3. Une fois cela terminé, toutes les feuilles issues du coup No 3 ont été examinées, et l'algorithme continue en s'intéressant dans l'ordre au coup No 1 (voir figure II-11) puis au coup No 2.

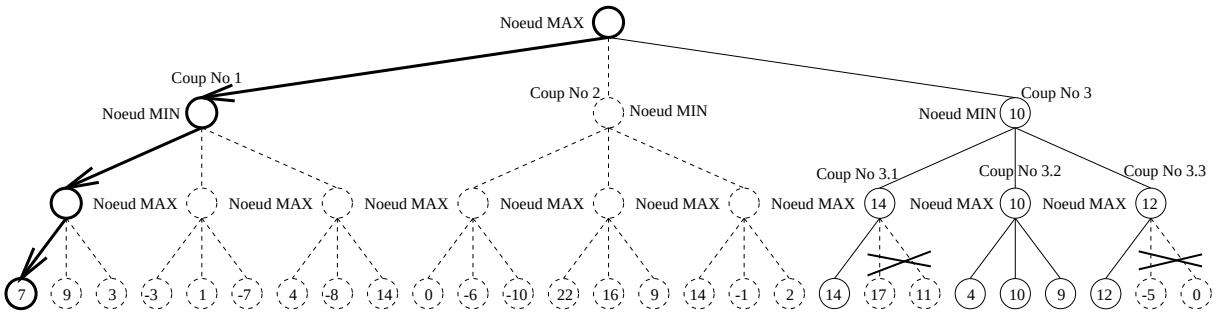


Figure II-11 : *Meilleure branche* : en cours de traitement

Le résultat définitif nous apparaît sur la figure II-12.

Quelques remarques suite aux résultats obtenus :

1. le meilleur coup trouvé reste bien sûr le coup No 3 ;
2. cette fois, le nombre de feuilles passées en revue est de 11 sur les 27 possibles, ce qui représente un gain de 2 feuilles sur la version avec juste *meilleur coup* ;

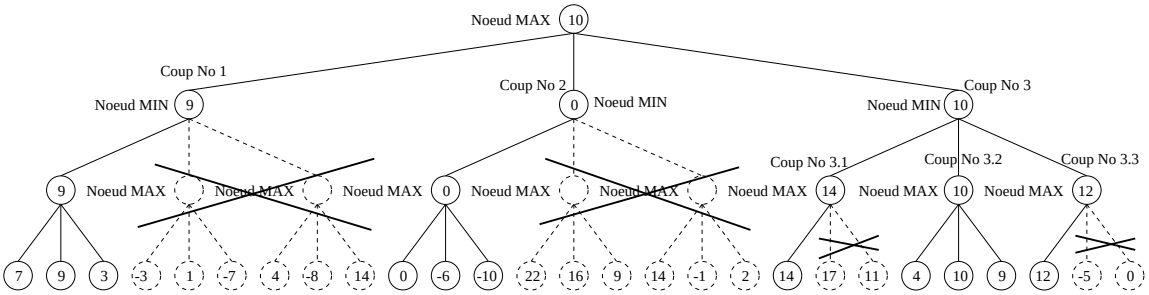


Figure II-12 : *Meilleure branche* : après traitement

3. on pourrait penser qu'un gain de 2 feuilles ne représente qu'une petite amélioration. Il n'en est rien ! Tout d'abord cela fait toujours 2 feuilles de moins à inspecter. Il faut savoir ensuite que l'exemple présenté est simplifié (facteur de branchement inférieur à la réalité, cas simples,...). Enfin, il ne faut pas oublier qu'en milieu de partie, OTAGE va à des profondeur de l'ordre de 12, 13 voire 14 coups : l'élagage induit est donc d'un tout autre ordre et très important (A titre d'information, se reporter au IV.1 pour une comparaison version de base / version améliorée)

Conclusion

Nous constatons que l'idée d'utiliser à une profondeur $d+1$ les informations que nous pouvons remonter à la suite d'une exploration de profondeur d, s'avère très fructueuse et mérite d'être mise en place. L'algorithme $\alpha\beta$ est d'autant plus efficace que l'on parcourt l'arbre "de la bonne façon". Quand le niveau est maximisant, il faut examiner d'abord les coups qui ont le plus de chance de générer des positions à forte valeur d'évaluation et réciproquement. En ce qui nous concerne, nous avons pour l'instant pensé à ramener la *meilleure branche*, mais voyons ci-après en II.3.2 si nous ne pouvons pas étendre un peu plus ce principe.

II.3.2 Tables de réfutation

Nous pouvons essayer d'exploiter encore plus d'informations et ne pas se limiter à n'utiliser que la *meilleure branche*.

Principe

Actuellement, nous ne ramenons que la branche concernant le meilleur coup, appelée *meilleure branche*. Nous pouvons améliorer cela en combinant l'approfondissement itératif avec des tables de réfutation[AN77] : lors de la recherche à la profondeur d, pour chaque mouvement de la racine, la meilleure variante de ce mouvement (variante qui mène au score *négamax*, même si celui-ci n'est qu'une borne) est stockée dans une table. La variante stockée pour le meilleur mouvement issu de la racine à la profondeur d est appelée la *variante principale* et n'est autre que ce que nous avons appelé jusqu'ici la *meilleure branche*. Les autres variantes montrent les suites de mouvements qui permettent à l'adversaire de "réfuter" les autres mouvements, c'est à dire rendre leur valeur *négamax* inférieure à la valeur *négamax* du meilleur mouvement. Lors de la recherche à la profondeur $d+1$, pour chaque mouvement de la racine, la variante correspondant à ce mouvement stockée dans la table sera essayée en premier.

Bien évidemment, nous commencerons par étudier en premier le coup correspondant à la *variante principale*, c'est à dire la *meilleure branche*.

Exemple

Nous allons à l'aide d'un exemple montrer l'élagage supplémentaire que nous apportent les *tables de réfutation*.

Partons donc d'un arbre de profondeur 2 que nous traitons par $\alpha\beta$. L'arbre utilisé est en fait le même arbre que celui de la figure II-3 qui a été utilisé jusqu'ici : nous ne faisons juste que permuter l'ordre de certaines feuilles. L'arbre avant traitement est présenté sur la figure II-13 et la figure II-14 nous montre le résultat.

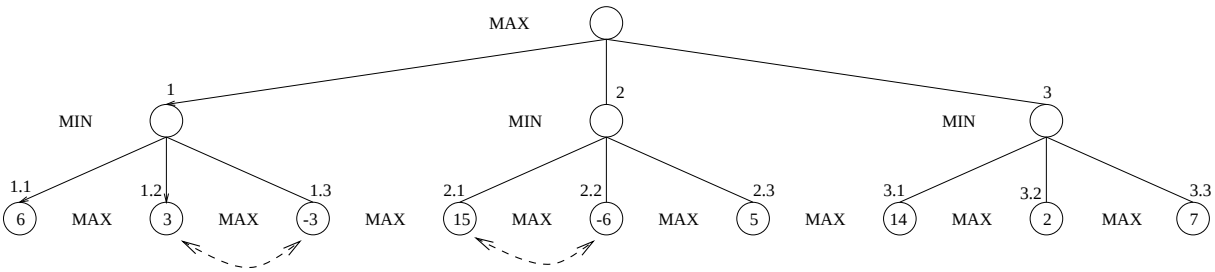


Figure II-13 : *Tables de réfutation* : profondeur 2, avant traitement

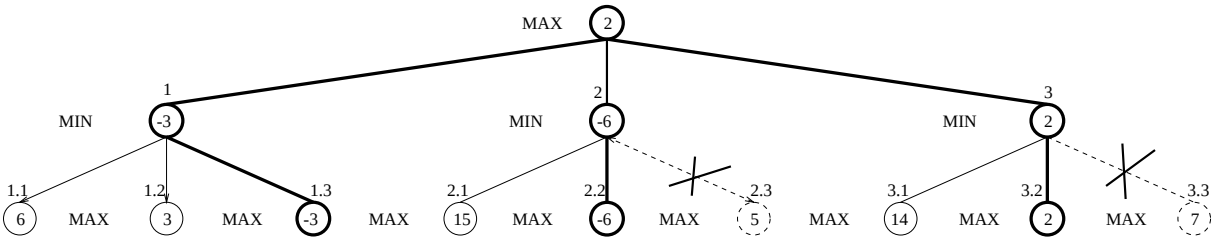


Figure II-14 : *Tables de réfutation* : profondeur 2, après traitement

Nous voyons que le coup retourné est le coup No 3. Les variantes sont les suivantes :

1. *variante principale* : $3 \rightarrow 3.2$;
2. première variante : $1 \rightarrow 1.3$;
3. deuxième variante : $2 \rightarrow 2.2$.

Lorsqu'on passe à la profondeur 3, l'arbre présenté sur la figure II-15 est parcouru en utilisant ces informations. On commence par étudier la *variante principale* : figure II-16. Une fois les feuilles issues du coup No 3 parcourues, on poursuit sur le coup No 1, en examinant sa variante d'abord : figure II-17. C'est ensuite au tour du coup No 2 d'être traité ; on passe bien sûr en revue sa variante en premier : figure II-18. Le résultat final est montré sur la figure II-19.

En fin de compte, le coup choisi est toujours le coup No 3 et nous n'explorons que 11 feuilles sur les 27 au total. A titre de comparaison, nous fournissons en annexe B les figures des résultats obtenus pour ce même arbre par l'intermédiaire des autres algorithmes vus

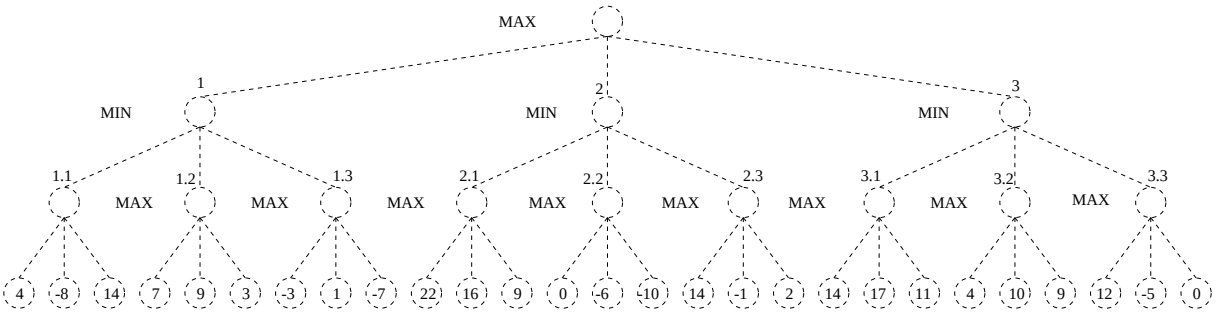


Figure II-15 : *Tables de réfutation* : profondeur 3, avant traitement

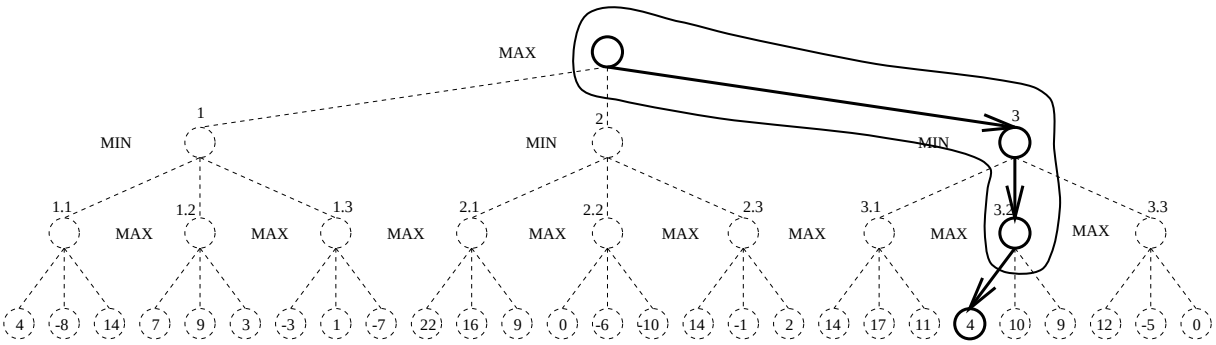


Figure II-16 : *Tables de réfutation* : *variante principale*

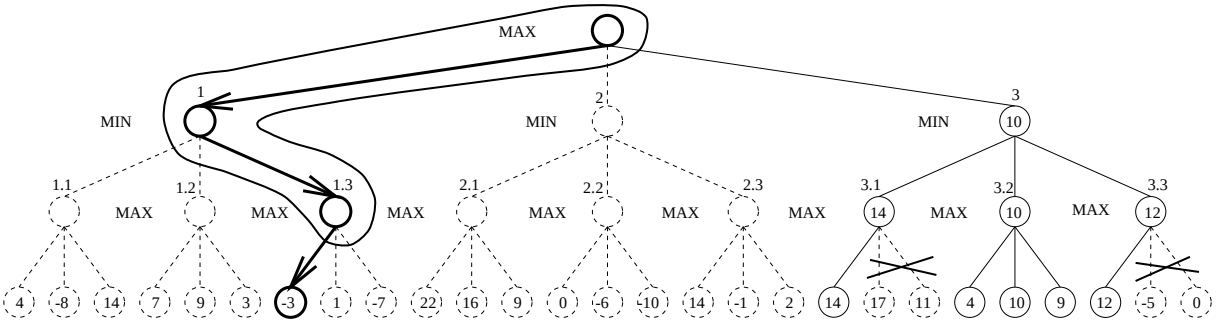


Figure II-17 : *Tables de réfutation* : première variante

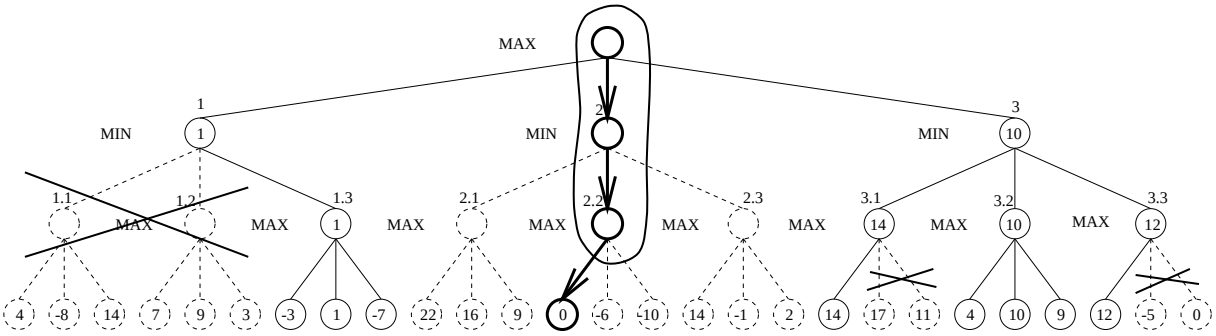


Figure II-18 : *Tables de réfutation* : deuxième variante

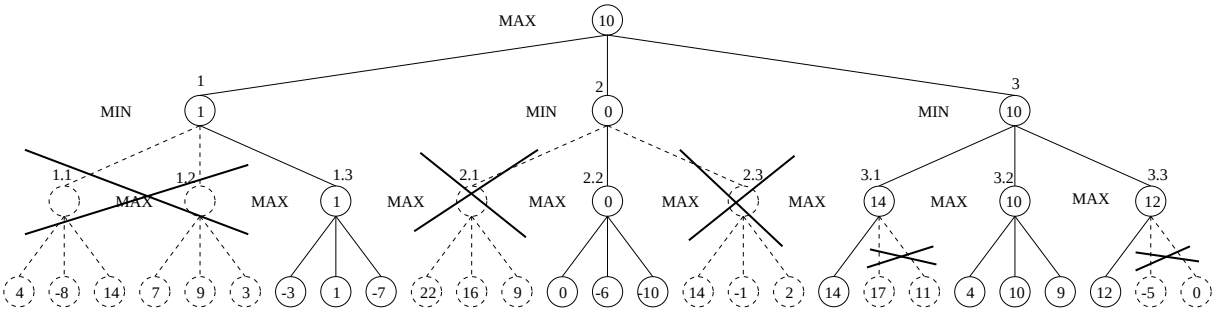


Figure II-19 : *Tables de réfutation* : résultat

jusqu'ici. Ci-après, nous trouvons un petit tableau récapitulatif montrant le nombre de feuilles parcourues de ce même arbre, pour chacun de ces algorithmes.

Type d'algorithme	nb / 27	%
minimax	27	100
$\alpha\beta$	20	74.1
$\alpha\beta$ avec *meilleur coup*	19	70.4
$\alpha\beta$ avec *meilleure branche*	17	63.0
$\alpha\beta$ avec *tables de réfutation*	11	40.7

Remarque sur l'implantation

Tous ces différents algorithmes représentent une amélioration du précédent. L'$\alpha\beta$ est un *minimax* avec élagage et l'on peut y ajouter le principe du *meilleur coup*. La technique de la *meilleure branche* n'est autre que celle du *meilleur coup* étendue à toute une branche. Enfin, les *tables de réfutation* stockent des variantes dont l'une d'elles, la *variante principale*, n'est autre que la *meilleure branche*.

Au vu de ces remarques, nous implanterons successivement l'$\alpha\beta$ avec *meilleure branche* et l'$\alpha\beta$ avec *tables de réfutation*. La structure développée pour ramener la *meilleure branche* pouvant être étendue et exploitée afin de mettre en œuvre les *tables de réfutation*.

II.3.3 Changement de l'ordre des coups

Nous imposons parfois quel coup il faut étudier en premier (cf. II.3.2), cependant, pour tout le reste du traitement par $\alpha\beta$, on étudie les fils d'un nœud du premier "à gauche" jusqu'au dernier "à droite" (voir figure II-20).

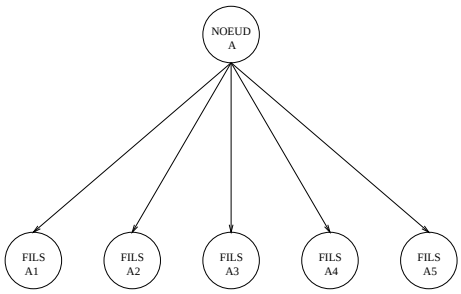


Figure II-20 : Ordre d'examination des fils d'un nœud

Sur la figure II-20, l'$\alpha\beta$ examinera les fils du nœud A dans l'ordre A1, A2 ,A3, A4 puis A5. Celui-ci correspond en fait à l'ordre dans lequel sont générés les fils du nœud A (voir I.2.2 / Principes de l'algorithme $\alpha\beta$ et I.2.2 / L'algorithme $\alpha\beta$). Dans OTAGE, cet ordre a été arbitrairement fixé d'une certaine manière, mais voyons s'il n'y a pas moyen d'en choisir un meilleur.

Toujours dans l'optique d'élaguer au maximum, il serait préférable de générer les fils d'un nœud en fonction de leur probabilité à induire des positions de forte valeur. Pour ce faire, nous allons classer les coups suivant la valeur stratégique des cases correspondantes sur le plateau. Nous avions déjà signalé par exemple que les coins du plateau de jeu sont d'une manière générale les cases les plus importantes. Poser un pion sur un des quatre coins engendre dans la majorité des cas une position nettement plus favorable.

Sur le plateau de jeu, nous pouvons distinguer les types de cases suivants, montrés sur la figure II-21.

	C	A	B	B	A	C	
C	X	F	G	G	F	X	C
A	F	D	E	E	D	F	A
B	G	E	○	●	E	G	B
B	G	E	●	○	E	G	B
A	F	D	E	E	D	F	A
C	X	F	G	G	F	X	C
	C	A	B	B	A	C	

Figure II-21 : Nomenclature des cases à OTHELLO

Les mouvements sont donc ordonnés suivant le type des cases[Fre80, Kie83, Wei89]. L'ordre de génération sera les coins, puis les cases A - D - B - E - G - F - C - X.

II.4 Les ouvertures

II.4.1 Nouvelle approche des ouvertures

Les résultats de l'étude menée grâce à la base de données des parties jouées (cf. II.1) montreront que la gestion des ouvertures n'est pas bonne. OTAGE se fait très vite sortir de sa bibliothèque d'ouvertures par les forts programmes. De plus, on peut constater sur le serveur I.O.S que les autres programmes continuent à utiliser leur bibliothèques d'ouvertures alors que OTAGE est déjà en mode "milieu de partie" depuis longtemps.

Il faut donc changer notre approche des ouvertures. Ceci est primordial ! Lorsque OTAGE sort vite d'une ouverture, c'est autant de temps gaspillé pour la recherche. Il va dépenser du temps pour répondre aux coups de son adversaire par $\alpha\beta$ alors que ce dernier ne dépense rien et continue à répondre instantanément puisqu'il est toujours dans ses ouvertures.

La qualité des ouvertures stockées dans le fichier récupéré sur INTERNET (voir II.1) est à mettre en doute. Le fait de sortir vite des ouvertures montre que le fichier est sûrement

incomplet. De plus, même si OTAGE ne sort pas tout de suite des ouvertures, la qualité de plusieurs de ces ouvertures est à mettre en doute puisque OTAGE se retrouve souvent en mauvaise posture juste après la phase d'ouverture de la partie. Ceci est dramatique, car malgré toutes les améliorations apportées à l'$\alpha\beta$ (voir II.3), le mal est déjà fait et est souvent irrécupérable. Enfin, certaines ouvertures stockées dans ce fichier sont connues pour être perdantes ou mauvaises.

Nous pensons donc utiliser pour les ouvertures, la base de données des parties déjà jouées. En effet, cette base qui recense les parties déjà jouées et leur résultat respectif peut constituer une énorme source de savoir et d'expérience pour OTAGE. Si cette base est suffisamment importante, OTAGE pourra s'en servir pour ouvrir de la meilleure façon qui soit.

II.4.2 Implantation des *coups interdits*

Nous pourrions aussi utiliser de concert avec cette nouvelle approche des ouvertures, la technique des *coups interdits* [Sch89a, Sch89b, Sch91]. Le principe est le suivant : si nous nous trouvons dans une position que nous avons déjà rencontré dans les parties précédentes, nous examinons quels coups ont déjà été répondus par OTAGE dans cette situation. Certaines de ces réponses ont peut-être amené une défaite : alors, avant de lancer l'$Iter\alpha\beta$, nous spécifions au programme la liste de ces mauvais coups afin qu'il ne les choisisse pas au terme de sa recherche. Si le coup choisi par l'$Iter\alpha\beta$ (qui est donc supposé être le meilleur coup à jouer) fait partie de cette liste de coups interdits, OTAGE devra en choisir un autre, quitte à ce qu'il soit estimé moins bon. Cela nous permet d'explorer les autres coups, même s'ils sont d'une valuation moins bonne et d'essayer d'autres variantes pour cette position, dans l'optique de trouver une parade à la mauvaise situation où nous nous trouvons : nous savons de toute façon que les coups déjà entrevus (les *coups interdits*) nous mènent vers une défaite probable.

II.5 Aspects techniques

II.5.1 Mise en place de l'autoplay

Nous nous orientons manifestement vers une utilisation beaucoup plus intensive des bases de données et notamment de celle des parties déjà jouées. Mais, pour que l'utilisation de cette dernière soit efficace et optimale, il faut que cette base soit très fournie et contienne donc un très grand nombre de parties jouées. Sinon, les informations que nous pourrions en tirer ne seraient pas pertinentes.

Mais augmenter cette base n'est pas évident. Chaque partie jouée sur le serveur I.O.S. dure entre 30 minutes et 1 heure : chacun des adversaires disposant de 30 minutes pour jouer tous ses coups. Même si les programmes rencontrés n'utilisent pas tous la totalité du temps qui leur est imparti, nous pouvons espérer faire à peu près seulement 24 parties par journée. De plus, toutes ces parties ne seraient pas intéressantes : cela dépend de la valeur de l'adversaire et du résultat de la partie.

Il serait donc très utile de pouvoir faire jouer OTAGE contre lui-même (i.e.. implanter un *autoplay*). Nous aurions un adversaire d'une valeur raisonnable (OTAGE lui-même !) et nous serions en mesure de jouer des parties de 3 minutes par joueur par exemple, au lieu des 30 minutes réglementaires. Nous pourrions donc jouer énormément de parties[1]. Nous allons

[1] approx. 24 x 60 / 3 = 1 440 / 3 = 480 parties par 24 heures

aussi pouvoir utiliser l'*autoplay* afin d'améliorer les ouvertures en explorant exhaustivement toutes les ouvertures possibles et d'une manière répartie, comme nous le verrons dans le chapitre suivant.

En revanche, l'utilisation de l'*autoplay* présente aussi quelques désavantages dont il faudra s'accomoder :

- le fait de jouer les parties en 3 minutes au lieu de 30 n'apporte pas la même qualité d'analyse. La version en 3 minutes n'explore pas les coups aussi profondément que si l'on jouait en 30 minutes ; les coups joués sont donc moins bons. Heureusement, OTAGE est classé premier sur le serveur I.O.S. pour ce qui concerne les parties en *Blitz*, c'est à dire pour les parties rapides. Le désavantage est donc atténué.

- L'autre désavantage vient du fait que OTAGE joue contre lui-même. Cela introduit un biais. Si OTAGE poursuit une séquence d'ouverture mauvaise, ou bien encore par exemple, s'il répond un mauvais coup, un coup perdant, et si par ailleurs il ne voit pas qu'il a très mal joué, OTAGE ne se corrigera pas et continuera d'explorer et de jouer ces séquences là... Il y a donc un risque de voir OTAGE s'obstiner à jouer et à développer plusieurs parties inintéressantes. Affronter d'autres programmes forts reste donc toujours une méthode très instructive.

Malgré ces deux points, il faut tout de même utiliser l'*autoplay* car les avantages apportés l'emportent sur les désavantages qui pourraient se manifester, que nous pouvons maîtriser en prenant les mesures adéquates.

II.5.2 Chargement automatisé dans la base de données

Actuellement, on ne rajoute des éléments dans la base de données des parties jouées que lorsque OTAGE joue sur le serveur I.O.S. ; cette maintenance est actuellement réalisée par le programmeur. Le serveur I.O.S. offre entre autres le service suivant : régulièrement, il envoie à tous les joueurs un mail des parties jouées. Nous récupérons ce mail et entrons les parties intéressantes dans la base de données en traitant ces fichiers.

Il est donc intéressant d'automatiser ce processus en permettant à OTAGE de stocker une partie dans la base de données dès la fin de celle-ci. On ne rajoutera que les parties perdues ou que l'on aurait dû perdre si l'adversaire n'avait pas fait une énorme bêtise, et les parties gagnées contre de forts adversaires (il n'est pas intéressant de garder la trace d'une partie que l'on a gagné face à un adversaire sans valeur).

De plus, l'ajout de ce mécanisme de stockage permet d'enregistrer aussi les parties qui ne sont pas jouées sur le serveur I.O.S et notamment les parties jouées contre lui-même en mode *autoplay* (voir plus bas en II.5.1).

Chapitre III

Réalisation

III.1 Exploitation de la base de données des parties jouées

Nous avons en effet développé, comme prévu, un petit logiciel pour manipuler et interroger la base de données des parties jouées. Mais comme ensuite ce module a été complètement intégré à OTAGE puis amélioré, nous en reparlerons dans la partie III.6.

III.2 Les *patterns*

III.2.1 Format des *patterns*

OTAGE fonctionne avec des *patterns* qui sont stockés dans des fichiers. Un fichier binaire recense l'ensemble des informations et c'est celui-ci qui est utilisé par le programme. Les autres fichiers (ASCII) permettent de voir les valeurs attribuées respectivement à chaque *pattern* concernant un bord, et à chaque *pattern* concernant un coin. Nous présentons ici le format et la signification de ces données.

Pour les bords

Voilà comment se présente une ligne du fichier concernant une *pattern* d'un bord :

```
1  1  0 -1 -1  1 -1 -1 =      2       4 0.500000
```

La signification de cette ligne est la suivante. La succession de 1, de 0 et de -1 décrit le *pattern* rencontré. Ici cela correspond au bord de plateau suivant :

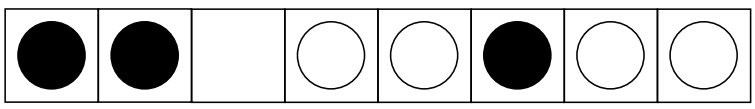


Figure III-1 : *Les bords* : image du *pattern*

Mais les *patterns* sont symétrisés (en position et en couleur) et ce même *pattern* stocké dans la base correspond en fait aussi à tous les autres *patterns* suivants :

Le chiffre 4 signifie qu'au cours de toutes les parties déjà jouées nous avons rencontré quatre fois l'une ou l'autre de ces *patterns*. Le chiffre 2 nous indique que sur ces quatre

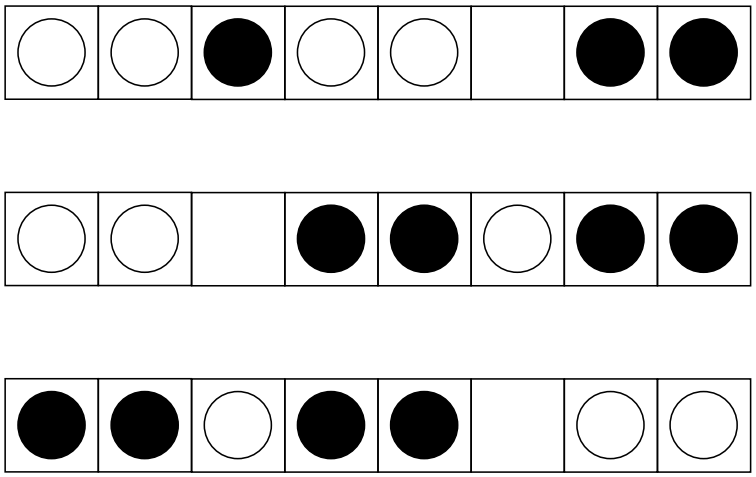


Figure III-2 : *Les bords* : autres *patterns* désignés

parties où nous avons rencontré ce *pattern*, la différence pour les Noirs des parties gagnées et des parties perdues. Enfin, le dernier nombre réel est tout simplement le rapport de ces deux chiffres.

Pour les coins

L'aspect d'un morceau du fichier concernant un coin est le suivant :

```
-1  1  1
-1 -1 -1
-1 -1 -1
=   -160     302 -0.529801
```

Le carré trois sur trois de 1 et de -1 nous décrit le *pattern*. Pour notre exemple, cela correspond au coin suivant :

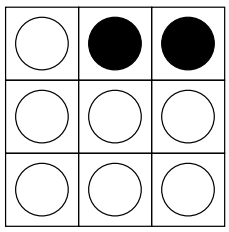


Figure III-3 : *Les coins* : image du *pattern*

Ce pattern est stocké dans le fichier du point de vue du coin en haut à gauche du plateau de jeu. Ce pattern correspond donc aux configurations suivantes sur les quatre coins :

De plus, comme précédemment, les *patterns* sont stockés symétrisés du point de vue de la couleur et de la position. Donc ce même *pattern* en désigne beaucoup d'autres et leurs corollaires sur les quatre coins du plateau de jeu. Nous montrons dans le schéma suivant

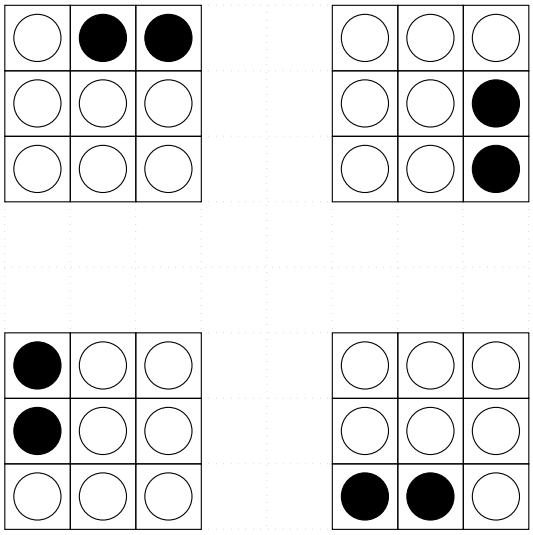
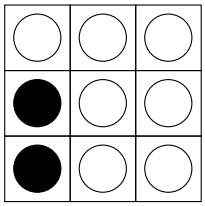


Figure III-4 : *Les coins* : image sur les quatre coins



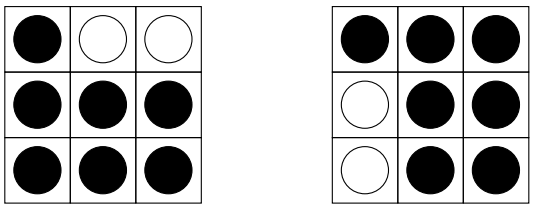


Figure III-5 : *Les coins* : autres *patterns* désignés

quels sont les autres *patterns* symétriques de celui stocké dans le fichier, pour le seul coin en haut à gauche (de simples rotations nous donnent les *patterns* pour les autres coins) :

Enfin, les chiffres désignent toujours la même chose, le nombre total de parties où l'on a rencontré ces *patterns* et parmi ceux-ci, la différence des parties gagnées et perdues par les Noirs. Le dernier nombre restant toujours le rapport de ces deux valeurs.

III.2.2 Prise en compte de la couleur qui va jouer

Nous avons rendu sensibles ces *patterns* à la couleur de celui qui va jouer (conformément aux remarques faites en II-2) : à cette fin, nous avons dupliqué tous les fichiers. Par exemple, en lieu et place du fichier pour les coins se trouvent maintenant un fichier pour ces *patterns*, lorsque les Noirs ont le trait, et un fichier de ces mêmes *patterns* lorsque c'est les Blancs qui vont jouer.

Au début, après duplication, ces deux fichiers jumeaux sont identiques et renferment la même information. Dans le traitement pour l'évaluation d'une position, nous prenons maintenant en compte la couleur de celui qui va jouer et le module qui se charge de l'évaluation d'une position a été modifié en conséquence. A l'initialisation du programme OTAGE , les fichiers sont chargés en mémoire, et lors de l'évaluation d'une position, OTAGE utilise pour un *pattern*, la valeur correspondante à ce dernier et correspondant à la couleur de celui qui va jouer.

III.2.3 Modification des *patterns*

L'après match

Lorsqu'une partie se termine, OTAGE commence tout d'abord par modifier plusieurs de ses paramètres (les $\alpha_1, \alpha_2, \ldots, \alpha_n$) en fonction du résultat de la partie. Dans un deuxième temps, OTAGE s'attache à ajuster et donc à modifier les valeurs de tous les *patterns* qu'il a rencontrés durant la partie. Nous sommes alors en mode *replay*. OTAGE procède comme suit...

Il passe en revue chaque *pattern* rencontré. Pour chacun de ceux-ci, il mémorise aussi la couleur de celui qui devait jouer au moment où ce *pattern* s'est présenté. Deux cas sont alors possibles. Ou bien OTAGE a gagné la partie, alors il va incrémenter la valeur de ce *pattern* s'il joue les Noirs, la décrémenter s'il joue le camp Blanc[1]. Inversement, si OTAGE a perdu la partie, il décrémentera la valeur du *pattern* s'il a joué les Noirs, et l'incrémentera s'il a joué les Blancs.

Ensuite, après avoir passé en revue tous les *patterns* du match et s'il a gagné la partie, OTAGE en reste là : l'analyse d'après match est terminée. C'est la fin du mode *replay* : OTAGE est prêt à jouer un nouveau match.

En revanche, si OTAGE avait perdu la partie, ce traitement n'est pas suffisant : il s'agit de voir si avec les valeurs modifiées des *patterns*, OTAGE jouerait autrement la même partie. Sinon, les modifications apportées n'auraient été d'aucune utilité. Supposons donc que OTAGE a perdu la partie. Il rejoue alors toute la partie (mais avec les nouvelles valeurs des *patterns*) : quand c'est à lui de jouer, OTAGE recherche sa réponse de la manière habituelle ($Iter\alpha\beta$), et quand c'est à son adversaire de jouer, OTAGE joue le coup que son opposant

[1]Nous rappelons que toutes les valeurs et tous les paramètres du programme sont stockés par rapport au camp Noir. Donc si OTAGE est Noir est qu'il a gagné, il faut bel et bien incrémenter le *pattern* pour signifier qu'il est favorable aux Noirs. Et vice versa si OTAGE joue les Blancs et qu'il a gagné la partie.

avait répondu[2]. Ce faisant, OTAGE regarde si, à un quelconque moment, il ne répond pas un coup différent de celui qu'il avait joué au cours de la partie originelle. Si c'est le cas, alors les modifications ont porté leurs fruits, et OTAGE ne tombera plus dans le même travers si cette même partie ou cette même position se présente à nouveau. C'est donc la fin du mode *replay*, OTAGE est fin prêt pour jouer à nouveau.

Mais si la partie se déroule de manière identique, alors il faut modifier encore les paramètres. OTAGE exécute alors une nouvelle fois toute la procédure d'après match. Il modifie ses paramètres, puis modifie encore de même les *patterns*. Ces derniers auront donc été incrémentés ou décrémentés deux fois. Et l'on poursuit ainsi : OTAGE rejoue encore la partie, si elle diffère, le traitement est fini, sinon, on modifie encore une fois les paramètres et les *patterns*. Ceci, autant de fois nécessaires pour que OTAGE ne joue plus la même suite de coups perdante.

Résultats

OTAGE cherchera comme toujours à obtenir pour les zones concernées du plateau (les bords et les coins), les *patterns* qu'il connaît être les meilleurs ; mais grâce à ces modifications, la stratégie d'OTAGE en ce qui concerne la prise des bords et des coins s'est modifiée et s'est affinée maintenant.

Nous pouvons remarquer que les fichiers jumeaux avec extension (dont nous avions dit qu'ils étaient identiques au début) diffèrent très vite après quelques parties, au niveau des *patterns* qui sont sensibles à la couleur de celui qui va jouer. Nous constatons que les valeurs de 10 *patterns* concernant les coins et de 9 *patterns* concernant les bords sont différentes dans les fichiers. Il était donc judicieux d'introduire cette nuance[3].

III.3 Récupérer les *variantes*

Suite à l'étude faite dans les parties II.3.1 et II.3.2, nous savons que grâce aux *variantes*, il est possible d'améliorer l'algorithme $\alpha\beta$, en lui permettant de parcourir beaucoup moins de feuilles pour un même résultat et ceci en moins de temps. Pour ce faire, nous allons expliquer tout d'abord comment nous ramenons la *meilleure branche* d'un arbre parcouru par $\alpha\beta$. L'implantation des *tables de réfutation* (c'est à dire des *variantes*) résultant du même principe (puisque, nous le rappelons, la *meilleure branche* n'est autre que la *variante principale*), toutes les structures de données et les outils nécessaires à leur implantation seront prêts.

III.3.1 Difficulté du problème

Ramener la *meilleure branche* pourrait sembler assez aisé. Une fois l'arbre parcouru, (voir figure III-6), il semble facile de pouvoir isoler la *meilleure branche* : *"il suffit de regarder"*. La *meilleure branche* est "évidemment" pour cet exemple : E6→ F4→D3 (voir figure III-7).

Mais le premier problème est qu'un algorithme de type $\alpha\beta$ ne garde pas la trace de l'arbre. Il ne fait que le parcourir, et c'est heureux car les arbres de jeux peuvent être

[2] OTAGE garde bien sûr la trace de chaque partie, et a donc l'historique des coups de la dernière partie jouée.

[3] Même si le nombre de ces *patterns* semble faible rapporté au nombre total de *patterns*, ces *patterns* font partie de ceux les plus fréquemment rencontrées durant les parties, et leur impact sur le jeu n'est donc pas négligeable.

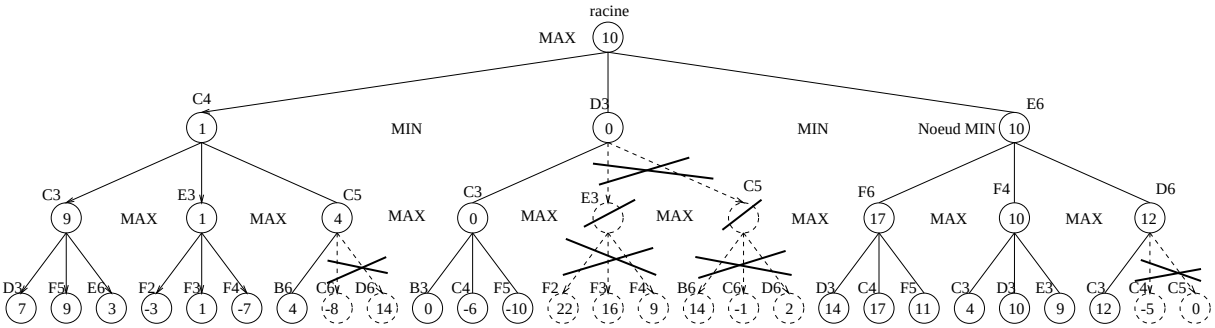


Figure III-6 : *variantes* : exemple d'un arbre traité par $\alpha\beta$

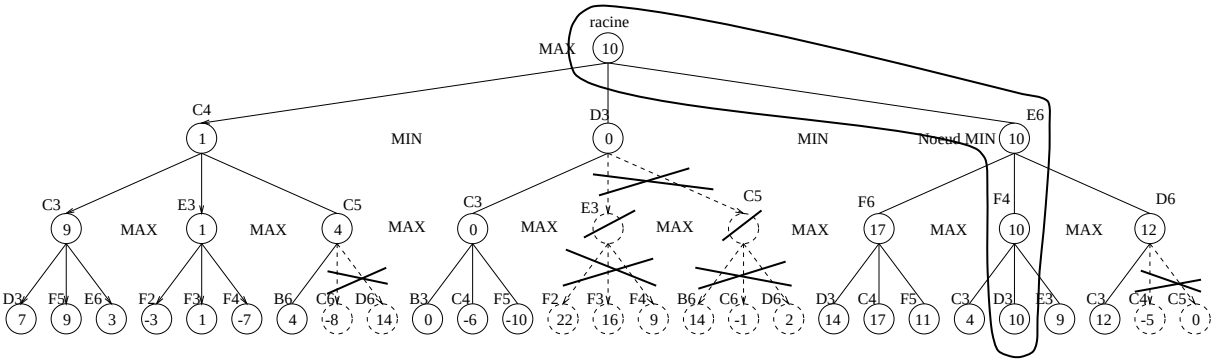


Figure III-7 : *variantes* : repérage de la *meilleure branche*

d'une taille énorme[4], voire souvent trop importante pour la mémoire de l'ordinateur. Par conséquent, nous ne savons pas à quoi ressemble l'arbre, après le traitement. Nous ne connaissons pas non plus l'aspect de l'arbre avant le traitement, puisque les branches de l'arbre sont générées au fur et à mesure qu'avance l'$\alpha\beta$ (voir I.2.2 / Principes de l'algorithme $\alpha\beta$ et I.2.2 / L'algorithme $\alpha\beta$). C'est donc au cours du traitement qu'il faut récupérer la *meilleure branche*.

L'autre difficulté réside dans le fait que l'$\alpha\beta$ parcourt l'arbre des feuilles vers la racine. Or la *meilleure branche* se construit dans l'autre sens : de la racine vers les feuilles (à titre de rappel, pour notre exemple, c'est E6→F4→D3).

Lorsqu'à une certaine profondeur, l'$\alpha\beta$ sélectionne un coup, c'est qu'il a trouvé, pour cette zone-ci de l'arbre, le meilleur fils pour cette profondeur[5]... Chaque fois que cela survient, nous pouvons essayer de garder dans un tableau (voir ci-dessus) l'identité du coup. Mais nous ne garderons pas un nœud élu suite à une coupure $\alpha\beta$, puisqu'il ne peut faire partie d'une variante[6]. Alors, nous aurons juste un tableau établissant une correspondance *profondeur* ↔ *meilleur coup* comme celui-ci :

[4]Nous rappelons que la progression du nombre de feuilles avec la profondeur de l'arbre est exponentielle.

[5]voir l'énoncé de l'algorithme en I.2.2 / L'algorithme $\alpha\beta$

[6]S'il y a eu coupure c'est que la valeur du nœud, de ses frères et des sous-arbres correspondants ne modifiera pas les bornes α et β de l'algorithme

Profondeur	Meilleur coup
0	la racine
1	...
...	...

Mais, nous pouvons remarquer qu'en fait, pour chaque profondeur, chaque fois qu'un nœud est élu, il écrase le coup précédemment stocké dans le tableau pour cette profondeur. Nous n'obtiendrons le bon résultat que si la *meilleure branche* que nous recherchons se trouve en "fin" d'arbre, c'est à dire "à droite" de l'arbre, en fin de traitement. Lorsqu'un coup est élu à une certaine profondeur, il ne faut donc pas écraser dans le tableau l'entrée précédente, mais plutôt conserver tous les nœuds qui ont été élus successivement pour une profondeur. Si l'on procède ainsi sur l'arbre de la figure III-8,

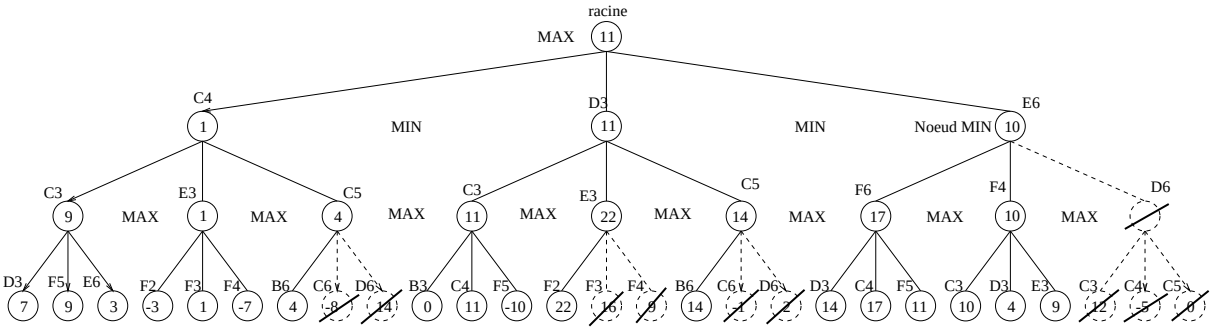


Figure III-8 : *variantes* : un autre exemple

on obtient alors le tableau suivant :

Profondeur	Meilleurs coups
0	la racine
1	**D3**
2	E3 **C3**
3	F5 F3 **C4** C4 C3

Cette fois nous avons toute l'information nécessaire... Si ce n'est que nous ne sommes plus en mesure d'extraire la *meilleure branche*. Nous connaissons toujours le *meilleur coup* (ici D3), donc nous savons que la *meilleure branche* commence ainsi : D3 $\rightarrow$?? $\rightarrow$?? .

Oui mais... Quel est le coup suivant ? E3 ou C3 ? Et c'est la même interrogation pour la profondeur 3...

III.3.2 La solution

La solution au problème qui consiste à ramener la *meilleure branche*, résulte des diverses constatations de la partie précédente III.3.1.

La dernière solution proposée apparaît intéressante. Il suffirait juste d'établir un lien entre un nœud père et ses fils. En fait, ce qu'il nous manque toujours, c'est la structure de l'arbre, comme nous l'avions remarqué au début de la partie précédente. Pour ce faire, nous allons marquer chacun des nœud que l'$\alpha\beta$ génère et ceci de manière *unique*. Nous pourrons utiliser par exemple un marquage à l'aide de nombres toujours croissants, 0 étant le numéro de la racine. De plus, chaque nœud généré recevra l'identificateur de son père. *Le*

lien sera ainsi créé. Dans le tableau, nous stockerons pour chaque coup, non seulement le coup lui-même, *mais aussi le numéro de son père et son numéro personnel.* Il sera ensuite très aisé, après le passage de l'$\alpha\beta$, de retrouver la *meilleure branche.*

Dans le tableau, nous utiliserons la notation suivante $\overset{12\rightarrow24}{C4}$ pour désigner le coup C4, dont le père a le numéro 12, et qui lui-même possède le numéro 24.

Si l'on en revient à l'arbre de la figure III-8, et que l'on numérote comme décrit plus haut tous les fils au fur et à mesure qu'ils sont générés (voir figure III-9), le tableau résultant

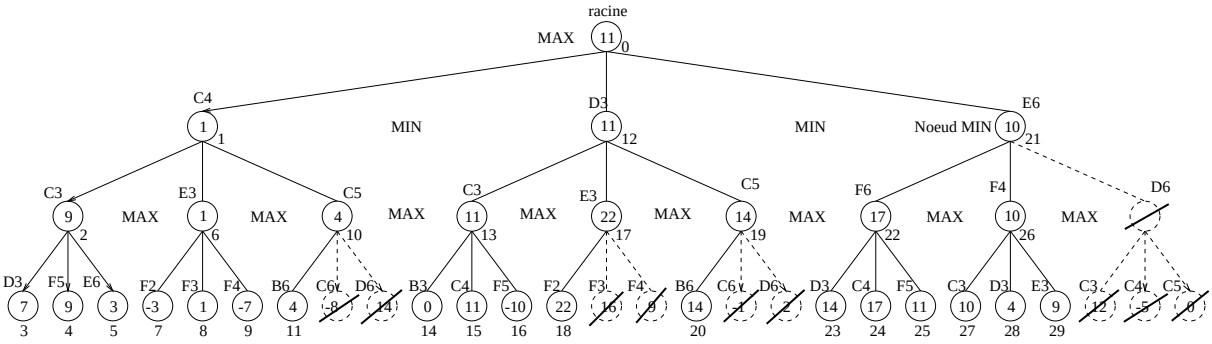


Figure III-9 : *variantes* : la solution

de la recherche de la *meilleure branche* devient :

Profondeur	Meilleurs coups
0	$\overset{\rightarrow0}{la\ racine}$
1	$\overset{0\rightarrow12}{\underline{D3}}$
2	$\overset{1\rightarrow6}{E3}\ \overset{12\rightarrow13}{\underline{C3}}\ \overset{21\rightarrow26}{F4}$
3	$\overset{2\rightarrow4}{F5}\ \overset{6\rightarrow8}{F3}\ \overset{10\rightarrow11}{B6}\ \overset{13\rightarrow15}{\underline{C4}}\ \overset{17\rightarrow18}{F2}\ \overset{19\rightarrow20}{B6}\ \overset{22\rightarrow24}{C4}\ \overset{26\rightarrow27}{C3}$

Il est alors aisé de retrouver la *meilleure branche.* On part de la racine, numérotée 0. Nous avons ensuite le *meilleur coup* D3 (qui bien sûr a pour père 0) et dont le numéro est 12. Pour trouver le coup suivant de la *meilleure branche* à la profondeur 2, il suffit de chercher parmi les meilleurs coups de profondeur 2 stockés dans le tableau, celui qui a 12 pour père. C'est le coup C3, numéroté 13. Enfin, nous obtenons le dernier coup de la séquence en procédant de même pour la profondeur 3 : cette fois le numéro à rechercher est le 13. Nous trouvons le coup C4. Finalement, nous avons bel et bien retrouvé la bonne suite de coups, celle qui forme la *meilleure branche* : D3→C3→C4.

III.3.3 Les *variantes*

Rappel

Les variantes ne sont autres que les séquences des meilleurs coups issues de chaque premier coup jouable de l'ordinateur. Ainsi, si l'on se réfère toujours au même arbre, celui de la figure III-9, le camp Noir (i.e. l'ordinateur) peut jouer trois coups : C4, D3 et E6. Si l'on récupère la suite des meilleurs coups issue de chacune de ses trois possibilités, nous obtiendrons toutes les *variantes.* L'une d'elles, celle qui commence par le meilleur coup à jouer pour les Noirs (ici D3) est appelée la *variante principale* et n'est autre que la *meilleure branche.*

Obtention des *variantes*

Nous pouvons déjà extraire la *variante principale* comme nous l'avons vu en III.3.2. Comme le tableau que nous utilisons pour récupérer cette *variante principale* recense tous les meilleurs coups pour chaque profondeur, il est très facile alors de générer les *variantes* non principales. Il suffit de faire figurer à la profondeur 1, dans le tableau, tous les coups jouables. En procédant de même que pour la *variante principale*, nous obtiendrons les autres *variantes*. Pour notre exemple, le tableau devient :

Profondeur	Meilleurs coups
0	$\overset{\rightarrow 0}{la\ racine}$
1	$\overset{0\rightarrow 1}{C4}\ \overset{0\rightarrow 12}{\underline{D3}}\ \overset{0\rightarrow 21}{E6}$
2	$\overset{1\rightarrow 6}{E3}\ \overset{12\rightarrow 13}{C3}\ \overset{21\rightarrow 26}{F4}$
3	$\overset{2\rightarrow 4}{F5}\ \overset{6\rightarrow 8}{F3}\ \overset{10\rightarrow 11}{B6}\ \overset{13\rightarrow 15}{C4}\ \overset{17\rightarrow 18}{F2}\ \overset{19\rightarrow 20}{B6}\ \overset{22\rightarrow 24}{C4}\ \overset{26\rightarrow 27}{C3}$

Nous avons souligné le *meilleur coup*. Par le même processus qu'à la partie précédente III.3.2, nous obtenons :

- variante principale : D3 $\rightarrow$ C3 $\rightarrow$ C4 ;
- première variante : C4 $\rightarrow$ E3 $\rightarrow$ F3 ;
- deuxième variante : E6 $\rightarrow$ F4 $\rightarrow$ C3.

III.3.4 Le rapport coût/gain obtenu

Une fois les *variantes* récupérées et utilisées dans le programme (voir III.4), nous pouvons constater que la gain est énorme : beaucoup moins de feuilles sont explorées et OTAGE gagne ainsi beaucoup de temps. On pouvait craindre cependant que le traitement associé au stockage dans le tableau des meilleurs coups, prenne trop de temps par rapport aux gains de temps apportés par les *variantes*.

Il n'en est rien. Il apparaît beaucoup plus avantageux de perdre un peu de temps pour récupérer les *variantes*, et les utiliser ensuite, plutôt que d'employer une version de OTAGE sans la technique des *variantes*. La perte de temps due à la collecte des *variantes* est largement compensée par les gains de temps réalisés lors l'itération suivante de l'$\alpha\beta$.

III.3.5 Un ennui majeur

Cependant, il subsiste un problème important. Comme nous l'avions fait remarquer, l'$\alpha\beta$ ne conserve pas la structure de l'arbre, ce qui est heureux car la taille de l'arbre peut être très grande. Mais c'est aussi cela qui rendait difficile la récupération des *variantes*. Du coup, nous avons utilisé les procédés décrits plus haut (III.3.1 et III.3.2) et à bien y regarder, ces techniques recréent quelque peu la structure de l'arbre, et c'est d'ailleurs bien ce que nous voulions.

En fait, pour être plus précis, nous recréons les parties de l'arbre qui représentent les meilleures suites de coups issues de chaque nœud. Nous pouvons assimiler ce processus au

traitement suivant : nous commençons par garder pour chaque ensemble de feuilles issues du même père, la meilleure de ces feuilles. Puis, passant à la profondeur inférieure, nous gardons parmi les branches déjà élues , la meilleure pour chaque ensemble de branches issues du même père. Nous procédons de cette manière ainsi de suite, pour arriver enfin jusqu'à la racine.

Nous allons montrer ce processus sur un exemple, toujours à partir du même arbre (celui de la figure III-9). Lorsque nous avons ramené ses *variantes*, le traitement revenait donc en fait à faire les opérations suivantes :

- garder les meilleures feuilles (figure III-10)

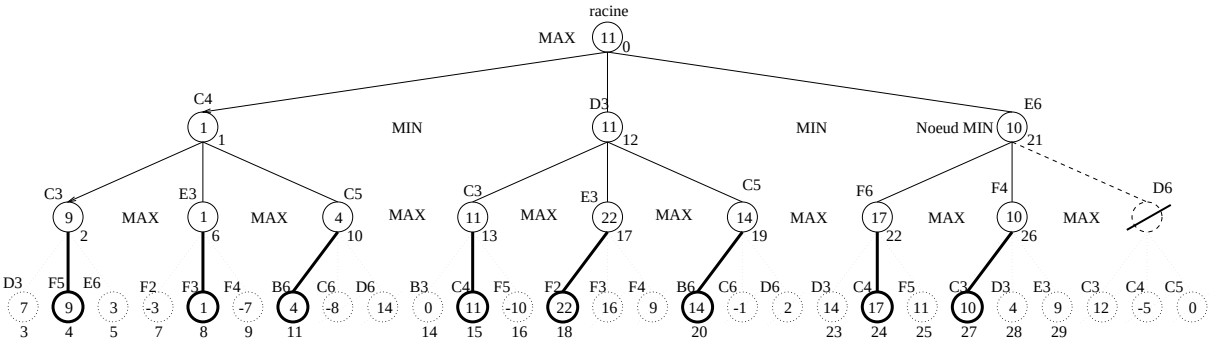


Figure III-10 : *variantes* : le processus, 1ère étape

- puis garder les meilleurs branches de ces feuilles-ci, pour la profondeur 2 (figure III-11)

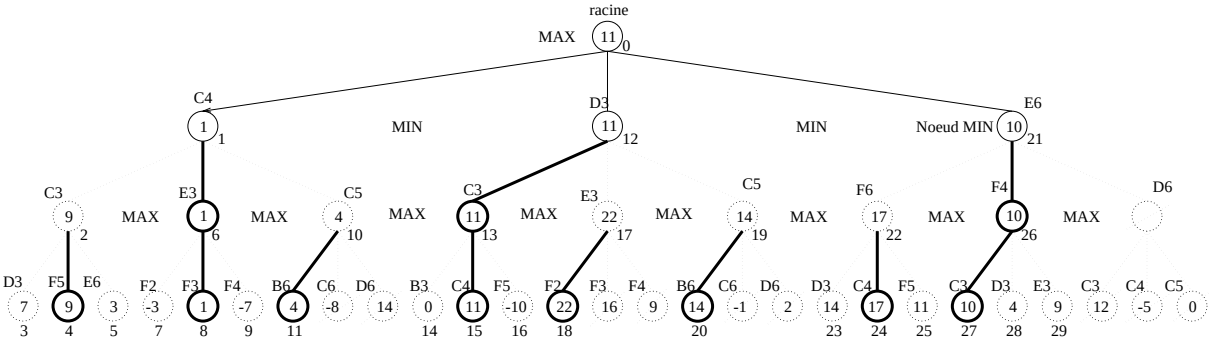


Figure III-11 : *variantes* : le processus, étape suivante

- et enfin garder parmi les branches restantes, la meilleure branche (figure III-12)

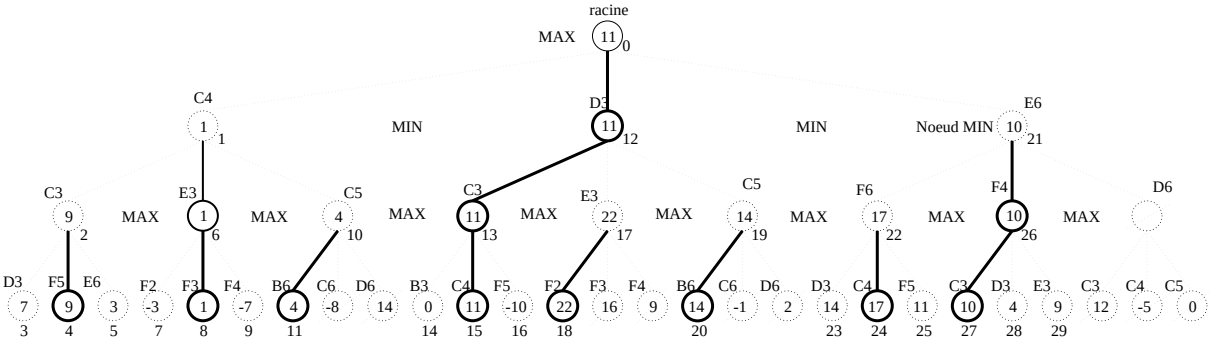


Figure III-12 : *variantes* : le processus, dernière étape

En définitive, nous avons remonté la structure que l'on présente sur la figure III-13 ci-après.

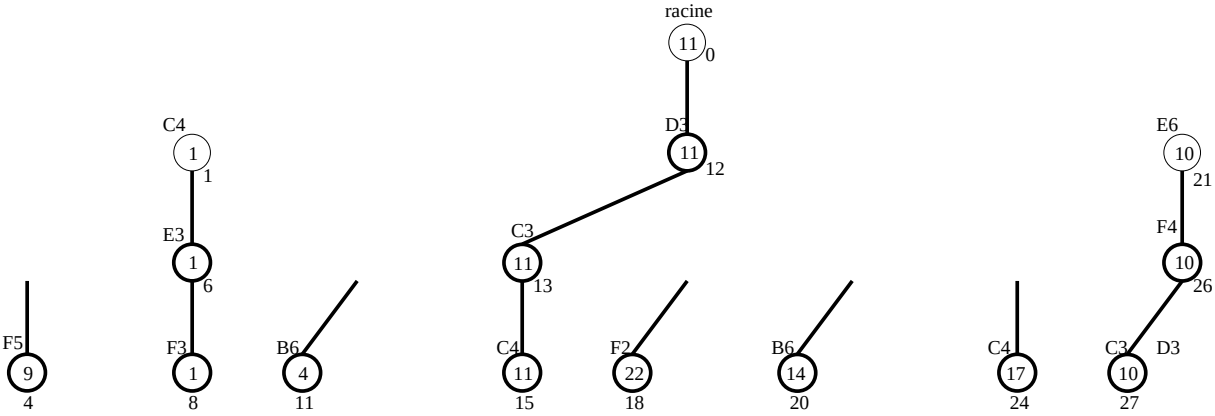


Figure III-13 : *variantes* : le processus, structure remontée

Le problème apparaît alors : nous gardons en fait en mémoire la structure de l'arbre au *facteur de branchement* près. En effet, pour tous les fils d'un nœud, nous gardons à chaque fois le meilleur.

De manière plus exacte, nous gardons toute la structure de l'arbre au *facteur de branchement* près, si l'arbre est entièrement parcouru (par un *minimax* par exemple). Si tel est le cas (traitement par *minimax*), le nombre de feuilles d'un arbre de profondeur p est 9^p puisque le *facteur de branchement* à OTHELLO est de 9. Donc, le nombre total de nœuds est :

$$1 + 9^1 + 9^2 + \cdots + 9^{p-1} + 9^p = \sum_{i=0}^{p} 9^i = \frac{1 - 9^{p+1}}{1 - 9} = \frac{9^{p+1} - 1}{8}$$

Le nombre de nœuds stockés alors dans le tableau est :

$$\frac{9^{p+1} - 1}{8 * 9} = \frac{9^{p+1} - 1}{72}$$

Si l'on applique ces formules à une profondeur de 12, on obtient un ordre de grandeur de 35 milliards ! ! Nous pouvons aussi, pour seulement donner un ordre d'idée, simplifier la formule précédente :

$$\frac{9^{p+1}-1}{8*9} > \frac{9^{p+1}-1}{9*9} \simeq \frac{9^{p+1}}{9^2} = 9^{p-1}$$

Ce qui pour notre exemple donne 9^{11} ! ! Il faut en plus, du point de vue informatique, multiplier cette quantité par la taille des données : le coup + le numéro du père + le numéro du coup.

Toutefois, pour un arbre parcouru correctement par un $\alpha\beta$, nous savons que seulement $2\sqrt{N}$ feuilles sont parcourues, où N désigne le nombre de feuilles total de l'arbre. Donc, le nombre de nœuds stockés dans le tableau sera, si l'on note $M = 2\sqrt{N}$, de l'ordre de :

$$\frac{M}{9^p} + \frac{M}{9^{p-1}} + \cdots + \frac{M}{9^2} + \frac{M}{9} = M(\frac{1}{9} + \frac{1}{9^2} + \cdots + \frac{1}{9^{p-1}} + \frac{1}{9^p})$$

soit :

$$M\sum_{i=1}^{p}\frac{1}{9^i} = M\frac{1}{9}\frac{1-\frac{1}{9^p}}{1-\frac{1}{9}}$$

$$= M\frac{1-\frac{1}{9^p}}{8}$$

ce qui donne :

$$= 2\sqrt{9^p}\frac{1-\frac{1}{9^p}}{8} = 2\frac{9^{p/2}-\frac{1}{9^{p/2}}}{8}$$

Nous pouvons faire l'approximation suivante (pour les grandes profondeurs) :

$$\simeq 2\frac{9^{p/2}}{8} = \frac{9^{p/2}}{4}$$

Toujours pour une profondeur de 12, cela donne $\frac{9^6}{4}$ ce qui est de l'ordre de 133 000 nœuds. Bien sûr, le gain par rapport au *minimax* est énorme, puisque nous avons un facteur de :

$$\frac{9^{p-1}}{\frac{9^{p/2}}{4}} = 4\frac{9^{p-1}}{9^{p/2}} = 4.9^{p/2-1}$$

Il n'en reste pas moins que le nombre de nœuds stockés reste grand ($\frac{9^{p/2}}{4}$) et peut atteindre le million pour une profondeur de 14, profondeur qui n'est atteinte que très rarement il est vrai, compte tenu du temps imparti pour jouer. Tout cet espace mémoire accaparé peut encombrer dangereusement la machine. Il va donc falloir y remédier.

III.3.6 Remarque importante

La recherche des variantes prend beaucoup de place ; mais en fin de compte, après traitement, les informations pertinentes se résument aux seules 9 *variantes*, si le *facteur de branchement* est de 9 comme nous le supposons toujours. Ces 9 *variantes* représentent au plus $9.p$ nœuds. Donc, la quasi totalité des nœuds stockés ne servent à rien[7]. Malgré tout, ils ont pourtant été indispensables à la génération de ces *variantes*.

[7]Fraction des nœuds utiles : $4.\frac{9.p}{9^{\frac{p}{2}}} = 4.\frac{p}{9^{\frac{p}{2}-1}}$. Avec $p = 12$ on a : $4.\frac{12}{9^5} = 8.13$ %

L'explication vient du fait qu'à chacune des profondeurs, nous stockons tous les meilleurs coups. Ceux-ci nous permettrons, à la profondeur inférieure, de poursuivre le tracé de ces variantes. Mais seuls quelques-uns d'entre eux se révéleront faisant partie d'une des variantes. Tous les autres nœuds que nous aurons stockés à cette profondeur seront inutiles. Mais bien sûr, nous ne le saurons que lorsque nous aurons traité la profondeur inférieure. Nous sommes donc en mesure, une fois la profondeur inférieure traitée, de déterminer les nœuds inutiles, et de les effacer du tableau : c'est ce que nous nous proposons de faire.

Revenons sur notre exemple de la figure III-9 pour étayer nos dires. Nous constatons en effet que dans le tableau résultat, que l'on rappelle ci-dessous, plusieurs nœuds ne servent à rien.

Profondeur	Meilleurs coups
0	$\overset{\rightarrow 0}{la\ racine}$
1	$\overset{0\rightarrow 1}{C4}\ \overset{0\rightarrow 12}{\underline{D3}}\ \overset{0\rightarrow 21}{E6}$
2	$\overset{1\rightarrow 6}{E3}\ \overset{12\rightarrow 13}{C3}\ \overset{21\rightarrow 26}{F4}$
3	$\overset{2\rightarrow 4}{F5}\ \overset{6\rightarrow 8}{F3}\ \overset{10\rightarrow 11}{B6}\ \overset{13\rightarrow 15}{C4}\ \overset{17\rightarrow 18}{F2}\ \overset{19\rightarrow 20}{B6}\ \overset{22\rightarrow 24}{C4}\ \overset{26\rightarrow 27}{C3}$

Ce sont, pour la ligne concernant la profondeur 3, les nœuds : $\overset{2\rightarrow 4}{F5}$, $\overset{10\rightarrow 11}{B6}$, $\overset{17\rightarrow 18}{F2}$, $\overset{19\rightarrow 20}{B6}$ et $\overset{22\rightarrow 24}{C4}$.

Nous allons montrer maintenant comment on en arrive à stocker des nœuds apparemment inutiles. Lorsque l'on commence le traitement par $\alpha\beta$, certaines feuilles sont donc stockées dans le tableau car elles représentent potentiellement des fins de séquence de *variantes* (voir III-14). Sur le schéma de la figure III-14, nous avons donc 3 feuilles qui viennent

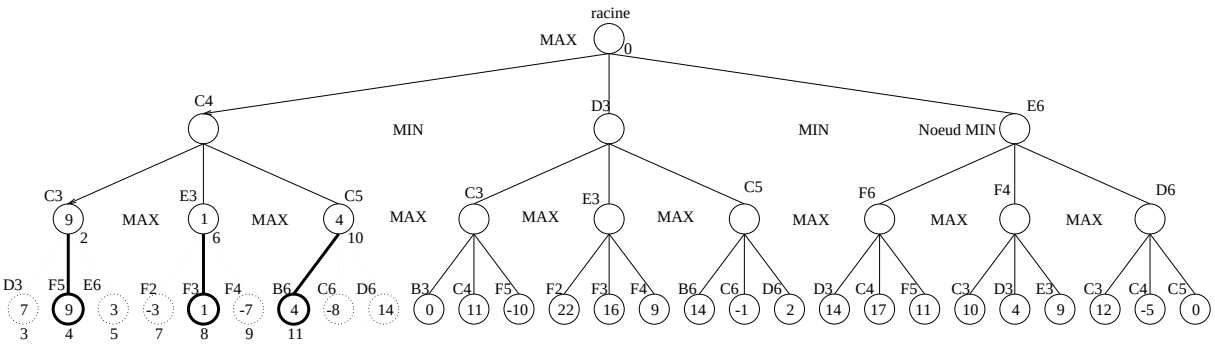


Figure III-14 : *variantes* : espace mémoire, début du traitement en profondeur 3

d'être élues : les nœuds numéro 4, 8 et 11. Le traitement par $\alpha\beta$ va se poursuivre, et remonter à la profondeur 2, où le nœud numéro 6 va être élu (voir figure III-15). Et c'est là que nous pouvons intervenir ! Car le fait d'avoir élu le nœud numéro 6 signifie que la séquence (nœud No 6) $\rightarrow$ (nœud No 8) a une chance de faire partie d'une variante ; mais surtout, cela condamne définitivement les nœuds 4 et 11 : ils nous seront d'aucune utilité et ne feront partie d'aucune *variante*. Rien ne sert donc de les garder dans le tableau. Ils correspondent d'ailleurs, bien évidemment, aux nœuds $\overset{2\rightarrow 4}{F5}$ et $\overset{10\rightarrow 11}{B6}$ que nous avions remarqué comme étant présents dans le tableau, mais pourtant fort inintéressants !

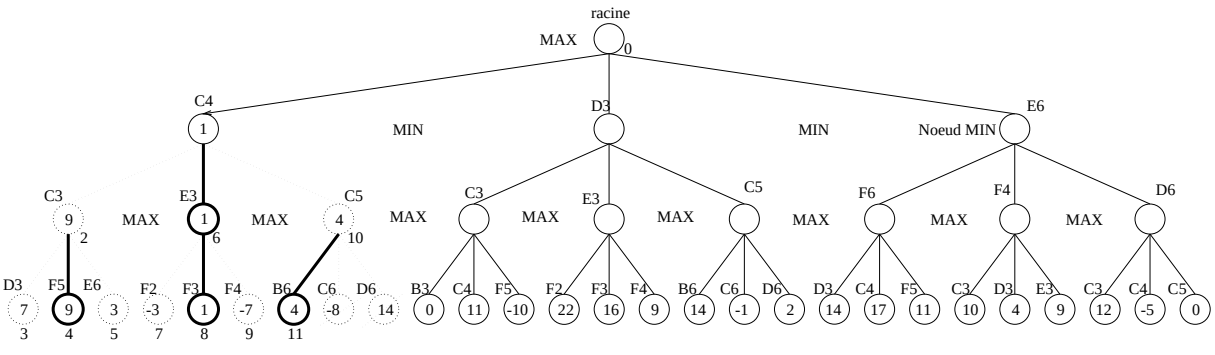


Figure III-15 : *variantes* : espace mémoire, poursuite du traitement en profondeur 2

D'une manière plus générale, chaque fois qu'à une profondeur nous élisons un nœud, nous pouvons retirer du tableau des nœuds de la profondeur supérieure qui maintenant ne servent à rien. Nous avons donc un moyen de nettoyer le tableau.

III.3.7 Libérer de la place mémoire

Nous sommes donc en mesure de libérer de la place mémoire, grâce à la remarque formulée ci dessus en III.3.6. Si l'on se rappelle de la proportion des nœuds utiles après traitement, par rapport au nombre de nœuds que nous stockons, nous voyons qu'un nettoyage du tableau libère quasiment toute la place occupée par celui-ci. Deux politiques ont été envisagées pour effectuer ce nettoyage.

Nettoyer en continu

Suites aux analyses faites précédemment, on peut penser nettoyer le tableau chaque fois que cela est possible, c'est à dire chaque fois qu'un nœud est élu et stocké dans le tableau (sauf si c'est une feuille). Nous nettoyons alors les données du tableau concernant la profondeur supérieure. Dans un premier temps, nous avons donc implanté et testé cette solution...

A tout instant du traitement de l'arbre de recherche par $\alpha\beta$, l'espace mémoire occupé était alors ridicule, et se résumait aux seules informations nécessaires à la génération des *variantes*. Mais du coup, OTAGE passait une majeure partie de son temps à nettoyer le tableau au cours du traitement par $\alpha\beta$. Par voie de conséquence, OTAGE perdait énormément de temps pour ce nettoyage, et il ne devenait plus avantageux d'utiliser les *variantes*. OTAGE perdait trop de temps à libérer de la mémoire, pour que l'utilisation des *variantes* à l'itération suivante lui soit bénéfique. Le rapport temps gagné grâce aux *variantes*/temps perdu pour la recherche des *variantes* lui était défavorable.

Nettoyer si nécessaire

Tout d'abord, il faut remarquer que le tableau , et toutes les autres données utiles au fonctionnement de OTAGE stockées en mémoire, ne généraient que très rarement des ralentissements dus au "swapping".

En fait, il suffit de nettoyer le tableau seulement lorsque la place occupée en mémoire par ce dernier atteint une certaine limite critique. Cette limite est un paramètre du programme OTAGE, et peut donc être fixée à toute valeur désirée. Ce faisant, OTAGE ne perd cette fois plus de temps à cause du nettoyage ; car comme nous l'avons remarqué, ce nettoyage n'est que rarement nécessaire. Lorsqu'il intervient quand même, il occasionne bien sûr un

surplus de temps, mais ce n'est donc pas souvent et pas systématiquement comme c'était le cas avec l'option *nettoyage en continu.*

III.4 Utiliser les *variantes*

III.4.1 Fonctionnement de l'$\alpha\beta$ d'OTAGE

OTAGE dispose d'un tableau représentant le plateau de jeu et d'une liste de coups recensant les 60 cases du plateau de jeu (toutes les cases moins les 4 premières cases du centre). Ces deux variables sont globales à tout le programme, pour des raisons d'optimisation.

Lorsque l'on appelle la procédure $\alpha\beta$, OTAGE parcourt cette liste des coups. Pour chacune des cases, il vérifie tout d'abord si elle n'est pas occupée et s'il peut jouer un pion sur cette case (conformément à la couleur de celui qui doit jouer), sinon il passe à l'élément suivant de la liste. OTAGE joue le coup et place donc sur la case correspondante un pion et retourne tous les autres pions qui doivent l'être. Ensuite, il poursuit le traitement par appel récursif à la procédure $\alpha\beta$. Lorsque le traitement associé à ce coup est terminé, OTAGE annule le coup : il enlève le pion de la case et remet les pions qui avaient été retournés dans leur position initiale. Ceci est nécessaire, car le plateau de jeu est une variable globale au programme : toute modification est globale au programme. Il n'y a pas de recopie du plateau de jeu chaque fois que l'on rappelle l'$\alpha\beta$. Après avoir annulé le coup, OTAGE continue à étudier les autres cases restantes de la liste des coups.

C'est ainsi que OTAGE procède pour effectuer son traitement. Il génère donc les coups dans l'ordre dans lequel ils sont stockés dans la liste des coups.

III.4.2 Rajout des *variantes*

La liste des coups dont nous venons de parler ci-dessus (cf. III.4.1), est déjà triée suivant un certain ordre. Il y a moyen de forcer OTAGE à explorer en premier lieu la *variante principale* puis ensuite les autres *variantes*, juste en touchant à cette liste. En effet, si la liste commence par exemple par les cases suivantes : C4 E3 F5 G6 ... Cela signifie qu'à l'appel de l'$\alpha\beta$, OTAGE va essayer en premier pour la profondeur 1, le coup C4. Si ce coup est valide, OTAGE va rappeler la procédure $\alpha\beta$ à la profondeur 2 : à ce stade, la première case de la liste est occupée, donc OTAGE va jouer le deuxième coup si celui-ci est valide, c'est à dire E3. A la profondeur 3, nous jouerons alors F5 (toujours si ce coup est valide).

Pour une liste se présentant sous la forme : [case1, case2, case3, case4, case5, ...], OTAGE va essayer en premier la séquence case1→case2→case3, si par exemple OTAGE travaille avec une profondeur de 3. Ainsi donc, nous sommes en mesure d'amener OTAGE à inspecter en premier les branches que nous voulons. Chaque fois que la procédure $\alpha\beta$ analyse un coup jouable *à la profondeur 1*, cela signifie que nous allons étudier toute la descendance de l'arbre issue d'un des premiers coups jouables de OTAGE , d'une de ses réponses possibles à l'adversaire. Mais, pour chacun de ces coups-réponse, nous possédons une *variante* qu'il convient d'explorer en premier.

Donc, au premier appel de la procédure $\alpha\beta$ *à la profondeur 1*, nous trierons la liste des cases de sorte à présenter en début de cette liste, la *variante principale* . Lorsqu'OTAGE aura fini le traitement associé au *meilleur coup*, et avant qu'il ne passe au coup suivant à la profondeur 1, nous trierons à nouveau la liste afin de lui présenter la *première variante.* Et ainsi de suite jusqu'à épuisement de toutes les variantes possibles.

Si l'on en revient à l'exemple de la figure III-9, nous avions obtenu les variantes suivantes :

- variante principale : D3→C3→C4 ;
- première variante : C4→E3→F3 ;
- deuxième variante : E6→F4→C3.

Si nous considérons maintenant l'itération suivante de l'$\alpha\beta$. L'arbre qui était de profondeur 3 va être maintenant exploré à la profondeur 4. Il devient alors comme le montre la figure III-16 sur laquelle ne figurer pas tous les nœuds de la profondeur 4. Supposons que la liste

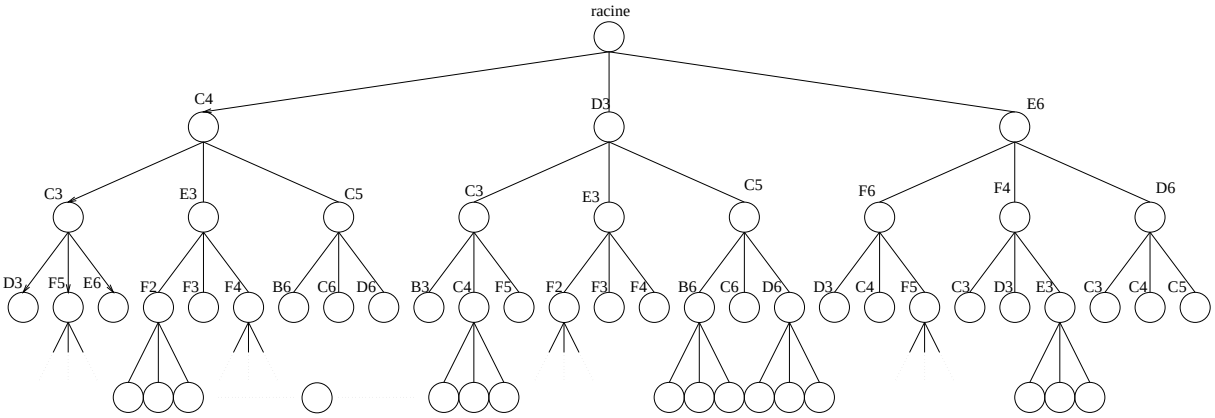


Figure III-16 : *variantes* : utilisation, l'arbre en profondeur 4

des cases soit initialement comme suit :

[A1, A2, A3, A4, A5, A6, A7, A8, B1, B2, ...].

OTAGE , muni de cette liste, commencera par étudier le coup C4, le premier dans l'ordre alphanumérique à être valide. En procédant ainsi, la première branche que OTAGE va observer sera : C4→C3→D3→premier fils de D3 par ordre alphanumérique. Puis il continuera ainsi et parcourra toute la partie de l'arbre associée au premier coup C4. Puis, OTAGE passera au coup D3, qui est le *meilleur coup* trouvé à l'itération précédente, puisque D3 est placé avant E6 dans la liste des cases. La branche issue de D3 qu'il étudiera ne sera pas la *meilleure branche* mais plutôt : D3→C3→B3→premier fils de B3 par ordre alphanumérique.

Si l'on trie la liste des coups, le traitement change et devient plus performant. A l'appel de l'$\alpha\beta$, on trie la liste suivant la *variante principale* et elle devient :

[**D3**, **C3**, **C4**, A1, A2, ..., A8, B1, ..., B8, C1, **C2**, **C5**, ..., C8, D1, **D2**, **D4**, ..., H8].

OTAGE va donc explorer en premier la *variante principale* suivie d'un des fils de C4, le premier â être rencontré dans la liste. OTAGE poursuivra son traitement et une fois la partie de l'arbre issue de D3 traitée, avant de poursuivre, nous trierons à nouveau la liste pour qu'il explore la *première variante*. La liste aura alors l'aspect suivant :

[**C4**, **E3**, **F3**, A1, A2, ..., A8, B1, ..., B8, C1, C2, **C3**, **C5**, ..., C8, D1, ..., D8, E1, **E2**, **E4**, ..., E8, F1, **F2**, **F4**, ..., H8].

Et ainsi de suite...

III.4.3 Précautions d'emploi des *variantes*

Il faut faire attention à deux points avant de décider de trier la liste des cases afin d'utiliser les *variantes*.

Premier point

Lorsque l'on réalise la *première* itération de l'$\alpha\beta$, c'est à dire lorsque l'on traite l'arbre pour la *première* fois, à la plus petite profondeur, nous ne disposons pas de *variantes* puisqu'il n'y a pas eu d'itération précédente. C'est cette itération que nous sommes en train de faire qui va générer les premières *variantes* qui seront utilisées lors de l'itération suivante.

Dans ce cas, nous utilisons la liste des cases telle qu'elle est initialement, sans la trier.

Deuxième point

Nous rappelons que notre $\alpha\beta$ (en fait $Iter\alpha\beta$) fonctionne à l'aide de *fenêtres aspirantes* dont nous avions parlé en I.2.2/ Fenêtres aspirantes. De ce fait, l'$\alpha\beta$ est susceptible, au cours d'une itération à une certaine profondeur, de s'arrêter et de se rappeler à la même profondeur (mais avec une fenêtre différente).

Ceci a une incidence sur la génération des *variantes*. Si un dépassement de fenêtre se produit, cela interrompt l'$\alpha\beta$ en cours de fonctionnement, mais surtout en ce qui nous concerne, cela interrompt la génération des *variantes*. Il ne faudrait pas qu'au cours de l'itération suivante (qui pourtant n'est autre que la même itération mais avec des données différentes pour l'$\alpha\beta$) nous essayions d'utiliser les variantes de l'itération précédente puisque celle-ci a été interrompue : les variantes ne seraient pas complètes, et ne concerneraient pas cette itération-ci de toute façon. Elles seraient destinées en fait à l'itération de profondeur suivante si cette dernière a lieu.

Pour remédier à ce problème, nous opérons comme suit. Lorsqu'une itération à une profondeur p vient de se terminer, le tableau utilisé pour la récupération des *variantes* vient d'être rempli. Admettons que OTAGE dispose de suffisamment de temps pour lancer l'itération suivante à la profondeur $p+1$. Avant d'effectuer à proprement parler cette itération à la profondeur $p+1$, nous générons, à l'aide du tableau précédent, les variantes v correspondantes et la nouvelle itération $(p+1)$ se poursuit en utilisant ces *variantes* v. Supposons maintenant que survienne un *dépassement*. Notre itération à la profondeur $p+1$ se termine donc en plein milieu de son traitement et la tableau qui stocke les nœuds pour les *variantes* est donc incomplet. Nous effectuons comme il se doit une nouvelle itération à la profondeur $p+1$. Mais avant qu'elle ne commence, nous devrions passer à la génération des nouvelles *variantes*. Si nous faisons cela, les *variantes* $v+1$ obtenues seraient fausses et erronées (à cause de l'imperfection du tableau), et qui plus est, ces variantes ne sont pas destinées à cette itération mais à la prochaine itération de profondeur $p+2$.

Pour éviter ces ennuis, nous détectons si il y a eu un *dépassement* en comparant tout simplement la profondeur actuelle de l'itération, avec la profondeur de l'itération précédente. Si elles sont toutes deux identiques, c'est qu'il y a eu *dépassement* et nous ne générons pas de nouvelles *variantes*, nous utilisons alors tout simplement, les anciennes variantes v.

Le déroulement du programme se poursuivra ainsi sans accroc. L'itération de profondeur $p+1$ se terminant sans problème, on effectue alors l'itération de profondeur $p+2$ sans oublier de générer les nouvelles *variantes* $v+1$ grâce au tableau qui cette fois est bien formé et bien complet.

III.5 Changement de l'ordre des coups

Comme nous l'avons vu au chapitre II.3.3, il est préférable de changer l'ordre des coups. Nous avons vu aussi dans la partie précédente III.4.2 comment se faisait exactement le

choix des coups à traiter en premier. En fait, le programme se base sur une liste de cases. Il traite donc les coups jouables dans l'ordre de cette liste. Mais nous avons vu aussi que nous retriions nous-mêmes cette liste afin de positionner en tête de liste les *variantes*. Est-il donc encore intéressant de vouloir imposer à cette liste un ordre particulier, si de toute façon elle doit être retriée pour les *variantes* ?

Nous répondons par l'affirmative à cause de deux raisons principales.

Première raison Lors de l'exécution de la *première* itération de l'$\alpha\beta$, nous ne disposons pas de *variantes*. Donc, la liste des cases ne sera pas retriée ; il est donc préférable d'avoir une liste avec un ordre pertinent, plutôt qu'un ordre quelconque.

Deuxième raison Si l'itération de l'$\alpha\beta$ courante utilise les *variantes*, elle a donc trié la liste afin de positionner en tête de liste la séquence de coups de la *variante* correspondante. Si l'on revient encore sur le même exemple, celui de la figure III-9. Nous avions trouvé comme variante principale D3→C3→C4. Observons l'itération suivante, sur l'arbre de profondeur 4 montré sur la figure III-16. Au début du traitement, nous avons donc trié la liste des cases de la sorte : [**D3**, **C3**, **C4**, ...]. La *variante principale* est en tête. Nous forçons donc indirectement les 3 premiers nœuds de la branche qui va être étudiée en premier. Mais nous sommes en profondeur 4 ! Il faut donc encore choisir un nœud parmi les fils du nœud C4 : D3→C3→**C4**→(?). Si nous laissons les choses telles qu'elles sont, OTAGE choisira parmi les fils de C4 le premier se présentant dans la liste des cases. Autant faire se peut, il vaudrait mieux que, parmi les fils, ce soit celui qui a la plus grande probabilité à induire des positions de forte valeur, comme nous l'avions vu en II.3.3. Nous voyons qu'il reste toujours intéressant de trier la liste. Mais la remarque est d'autant plus valable lorsque OTAGE est sorti de la *meilleure branche*. Même s'il explore en premier la *meilleure branche* et si cela engendre des coupures, OTAGE doit quand même visiter toutes les autres branches non coupées issues du coup C4. Et là, nous ne forçons plus aucun coup. Si la liste des cases est triée de manière pertinente, OTAGE visitera alors en premier les meilleurs nœuds.

Nous trions donc par défaut la liste dans l'ordre suivant que nous avions énoncé en II.3.3 :

[H8,A8,A1,H1,
A3,A6,C1,F1,C8,F8,H3,H6,
C3,C6,F3,F6,
D1,E1,A4,A5,D8,E8,H4,H5,
D3,E3,C4,C5,F4,F5,D6,E6,
D2,E2,B4,B5,D7,E7,G4,G5,
B3,C2,B6,C7,G6,F7,G3,F2,
A7,A2,G1,H2,B1,B8,H7,G8,
B2,G2,B7,G7]

III.6 Les ouvertures

Comme nous l'avions diagnostiqué, OTAGE possède une mauvaise bibliothèque d'ouvertures, et n'est pas très bon dans cette première phase d'une partie d'OTHELLO. Nous allons décrire les améliorations de cette partie du jeu que nous avons tenté de mettre en place.

III.6.1 La base de données

Nous disposons d'une base de données des parties déjà jouées par OTAGE. Cette base contient environ 300 parties. Nous avons développé dans un premier temps, un programme destiné à nous présenter les statistiques de ces parties. Ce programme classifie les parties en un arbre des coups joués comme le montre la figure II-1. Mais en fait, chaque nœud est une structure collectant les informations suivantes :

- identité du coup
- nombre de parties jouées ayant suivies cette séquence
- nombre de parties gagnées
- nombre de parties perdues
- somme algébrique des pions gagnés et perdus

Nous avons constaté les problèmes suivants :

- OTAGE joue seulement les ouvertures contenues dans le fichier mais du coup il n'exploite pas d'autres ouvertures que l'on sait être "bonnes" : le fichier des ouvertures est incomplet
- OTAGE utilise certaines ouvertures qui sont "mauvaises"
- OTAGE perd beaucoup de parties contre de forts programmes à cause de cette faiblesse chronique dans les ouvertures

III.6.2 Une nouvelle approche

Nous avons tout d'abord essayé de trouver un autre fichier sur les ouvertures à OTHELLO et de meilleure qualité. Le problème, c'est que de tels renseignements ne sont pas divulgués et nous n'avons pu obtenir ces informations. En effet, les divers auteurs connus des très bons programmes d'OTHELLO gardent chacun jalousement leurs techniques d'ouvertures, puisqu'il y a compétition entre les divers programmes, et ma foi.

Nous avons alors décidé d'utiliser l'ensemble des parties jouées par OTAGE comme une bibliothèque d'ouvertures. En effet, ces parties déjà jouées renferment divers enseignements et toute l'expérience d'OTAGE, et il serait regrettable de ne pas exploiter ces informations qui pour l'instant ne servaient qu'à nous fournir des statistiques sur les parties jouées.

III.6.3 Le principe

Génération de l'arbre

Voici comment nous allons procéder... A l'initialisation de OTAGE, le fichier des parties déjà jouées est parcouru, et l'on construit un arbre des séquences de coups de ces parties (sur le même principe que celui de la figure II-1). Nous possédons alors un arbre des parties déjà jouées, dans lequel toute redondance est supprimée : deux parties commençant par les mêmes coups sont stockées issues de la même branche. On ne répète donc pas les mêmes séquences de coups ; seules les divergences sont stockées. On se contente juste d'incrémenter pour les nœuds concernés par cette redondance le compteur de parties jouées et suivant le

cas celui des parties gagnées ou perdues, ainsi que la somme algébrique des pions gagnés et/ou perdus.

De plus, nous rappelons que le plateau de jeu présente diverses symétries. Dès lors, certaines séquences de coups qui pourraient apparaître différentes entre elles, concernent en fait les mêmes ouvertures (à une symétrie près). Nous allons donc symétriser l'arbre afin d'exploiter à son plus haut rendement les informations contenues dans le fichier des parties déjà jouées. Tous les premiers coups des parties seront par convention C4. A titre d'exemple, observons la figure II-1 qui nous montre un arbre *non symétrisé* des coups des parties jouées. La quatrième partie (E6 F4 C3 C4 D3 ...) ne commence pas par C4 : pour la stocker dans notre *arbre symétrisé*, nous allons la "symétriser" par rapport à la diagonale du plateau de jeu (voir tableau ci-dessous). Sur cette même figure, nous remarquons à l'occasion que cette séquence de début de partie (la quatrième), une fois symétrisée, n'est autre que la deuxième séquence stockée dans l'arbre : d'où l'intérêt de symétriser.

Nature de la partie	1er coup	2ième coup	3ième coup	4ième coup	5ième coup	...
Partie d'origine	E6	F4	C3	C4	D3	...
Partie symétrisée	C4	E3	F6	E6	F5	...

Si l'on représente sur un plateau de jeu les deux suites de coups précédentes, nous observerons bel et bien la même position sur le plateau de jeu, mais à une symétrie près (voir figure III-17).

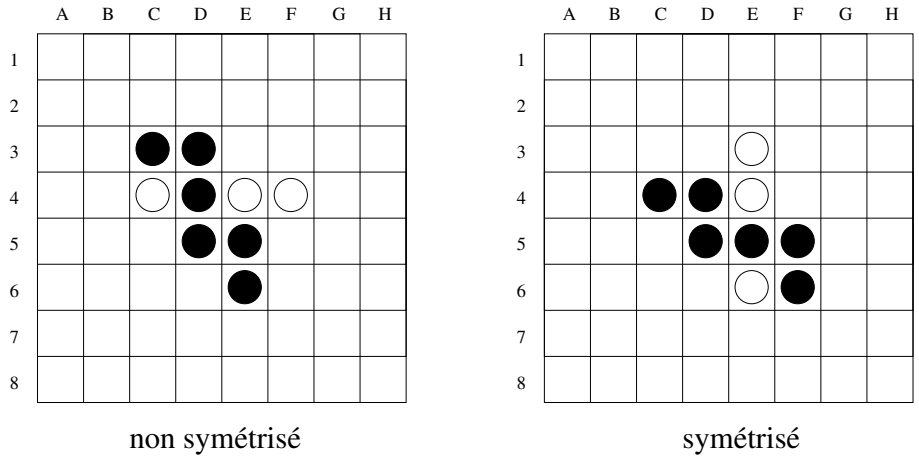


Figure III-17 : *ouvertures* : comparaison symétrisée / non symétrisée

L'arbre s'en voit encore simplifié, et celui de la figure II-1 prend alors la forme définitive sous laquelle nous allons l'exploiter (figure III-18).

Enfin, pour cette arbre que nous allons utiliser, nous n'avons pas besoin des statistiques ayant trait au nombre de parties jouées ni au nombre de parties gagnées ou perdues. Nous stockons seulement au niveau des feuilles de l'arbre (qui sont donc les derniers coups de chaque partie jouée) le différence de pions de cette partie. Un nombre positif signifie une victoire, un nombre négatif une défaite, et zéro un match nul, pour les Noirs.

Interprétation de l'arbre

Nous avons donc un arbre de coups symétrisé dont les feuilles sont évaluées et qui a donc l'aspect montré sur la figure III-19. Nous rappelons que les évaluations sont toujours faites

Exemple trivial du l'aspect général du stockage dans la base du début de 5 parties jouées

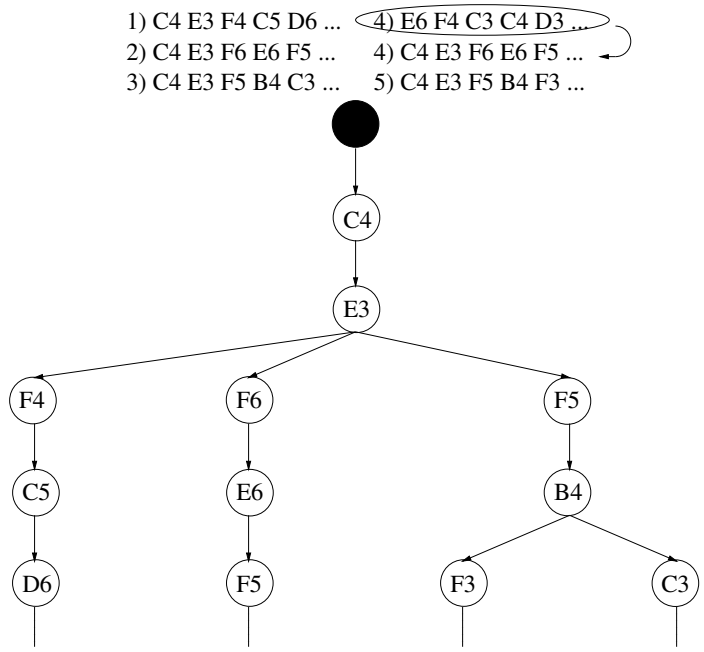


Figure III-18 : *ouvertures* : base de données symétrisée

par rapport aux Noirs. Sur un tel arbre, nous pouvons dire que OTAGE a joué 14 parties pour l'instant (c'est à dire le nombre de feuilles, à moins que OTAGE ait eu à jouer deux fois la *même* partie). Nous savons que sur toutes ces parties, les Noirs ont à la fois perdu et gagné. Il y a même eu un match nul, dont le dernier coup fut A1. Les parties jouées ont commencé soit par C4 C3, C4 E3 ou C4 C5. Une branche *complète* de l'arbre, à partir de C4 et jusqu'à une feuille, correspond à tous les coups d'une seule et même partie. Nous pouvons remarquer que chaque nœud d'un tel arbre n'a qu'un père et un seul.

De plus, si l'on considère un nœud particulier de l'arbre (prenons à titre d'exemple celui de la figure III-20), nous pouvons en tirer les informations suivantes :

- le nœud E6 représente toutes les parties jouées ayant suivi la même séquence depuis le premier coup C4 pour aboutir à ce coup. En l'occurrence ici, c'est la séquence : C4 C3 C2 C5 E6.
- Après cette séquence de coups, nous savons qu'au moins trois variantes ont été jouées :
  1. C4 C3 C2 C5 E6 **F6**
  2. C4 C3 C2 C5 E6 **D6**
  3. C4 C3 C2 C5 E6 **B2**
- A titre de remarque, nous pouvons même noter qu'au moins 6 parties ont suivi la séquence C4 C3 C2 C5 E6 si l'on regarde les branches qui partent des fils F6, D6 et B2.

Remarque très importante Lorsque l'on observe le nœud E6, on serait tenté de penser qu'il y a 3 réponses possibles à la séquence C4 C3 C2 C5 E6 (les réponses F6, D6 et B2).

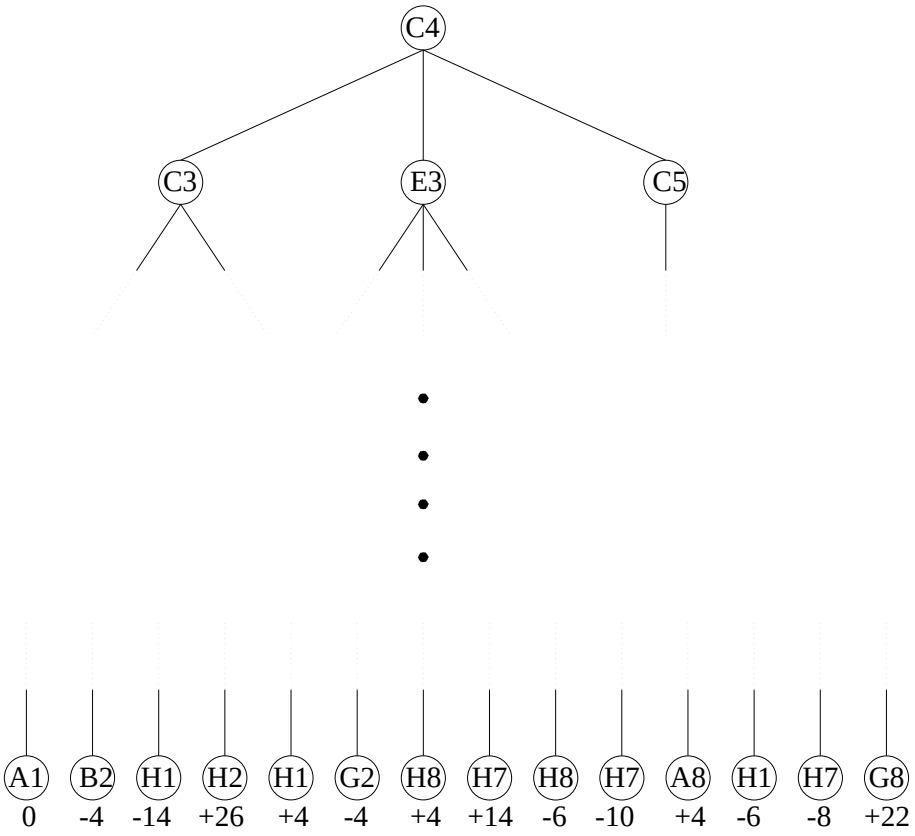


Figure III-19 : *ouvertures* : un exemple d'arbre des parties jouées

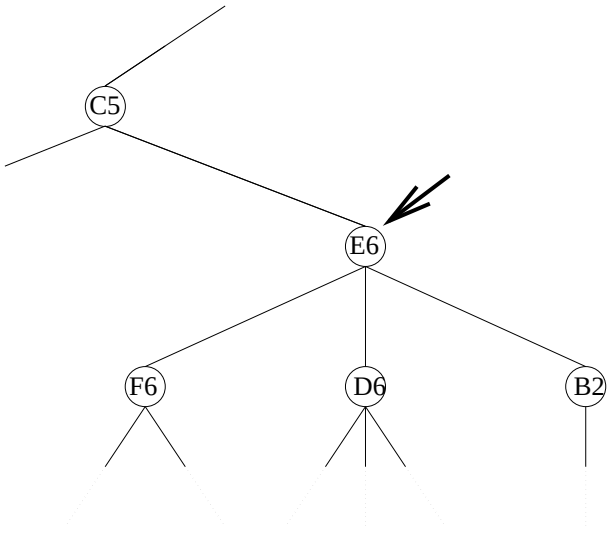


Figure III-20 : *ouvertures* : examen d'un nœud de l'arbre

Il n'en est rien ! Nous le rappelons, ce n'est pas un arbre exhaustif de toutes les séquences possibles à OTHELLO : cette arbre ne recense que les séquences que OTAGE a *effectivement* jouées. D'ailleurs, suite à la séquence de coups C4 C3 C2 C5 E6, il existe **7** réponses possibles (C1, F6, F4, F5, D6, B4 et B2) parmi lesquelles on trouve bien sûr F6, D6 et B2.

Évaluation préliminaire de l'arbre

Avant que OTAGE ne commence à jouer, il applique à cet arbre un *minimax* : nous rappelons que les feuilles ont chacune une valeur. Ceci fait, chaque nœud de l'arbre est maintenant libellé par une valeur qui représente la valeur *minimax* du sous-arbre dont il est la racine. Nous maximisons les profondeurs correspondant au choix d'un coup des Noirs, et nous minimisons pour les Blancs, comme de coutume, car les feuilles sont évaluées du point de vue des Noirs.

Grâce à l'application de ce *minimax*, la valeur de chacun des nœuds de l'arbre nous indique quelle quantité de pions nous pouvons espérer gagner (ou perdre...) à partir de cette séquence. Cela nous aidera pour le choix de la réponse à jouer, puisque nous avons l'évaluation des fils de chaque nœud. Revenons à l'exemple de la figure III-20 et évaluons l'arbre : nous obtenons la figure III-21 (où les valeurs ont été mises au hasard : elles ne sont pas réelles). OTAGE joue les Blancs, et nous avons assisté à la séquence de coups suivante :

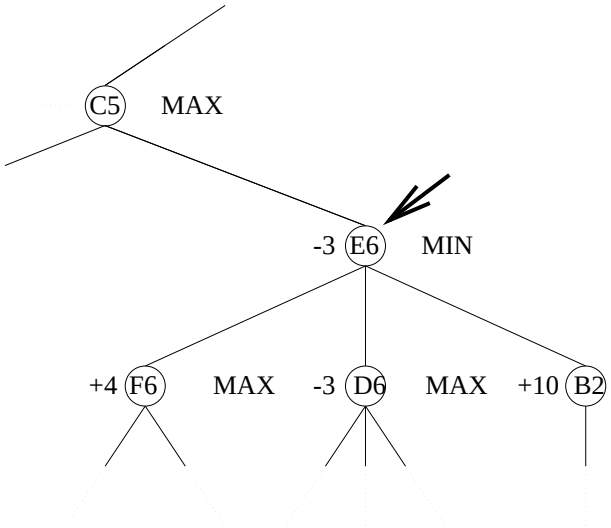


Figure III-21 : *ouvertures* : examen d'un nœud de l'arbre *évalué*

C4 C3 C2 C5 E6. C'est maintenant aux Blancs donc à OTAGE de jouer. La valeur -3 qui a été remontée au nœud E6 nous indique qu'à ce stade, les Noirs vont perdre au moins 3 pions, donc OTAGE en gagner 3. Il "suffit" pour cela à OTAGE de répondre D6.

Remarque très importante Mais cette valeur de -3 reflète-t-elle vraiment la réalité ? Encore une fois, cette valeur est le fruit d'un *minimax* appliqué à un arbre incomplet. La valeur -3 signifie plutôt : "sur toutes les parties déjà jouées et commençant par cette séquence, nous pouvons dire qu'à ce stade, les Noirs perdront au moins 3 pions". Si telle est vraiment la signification de cette valeur, alors elle peut nous paraître sans intérêt : qui nous dit que les futurs coups de la partie courante vont suivre la même séquence qu'une des parties déjà jouées ? Qui nous dit que l'adversaire ne va pas répondre différemment et emprunter une branche encore inconnue de l'arbre ?

Et pourtant, il ne faut pas oublier que nous ne stockons que les parties jouées contre des adversaires de très haut niveau. Les réponses enregistrées dans la base de données sont les meilleurs coups répondus par des programmes très forts. Donc, il y a de fortes chances pour que si OTAGE réponde D3 (pour notre exemple), l'adversaire n'ait comme meilleure réponse qu'une de celles stockées dans l'arbre : celle qui minimise ses pertes à "seulement" 3 pions. D'un autre côté, il peut arriver aussi qu'une réponse autre que celles prévues soit meilleure pour l'adversaire, et qu'il la joue : peut-être parce que cet adversaire aura exploré plus profondément que ne l'avaient fait jusque là les programmes rencontrés, ou bien parce que sa fonction d'évaluation estime différemment de celles des autres programmes la position ou pour bien d'autres raisons obscures encore...

Cette valeur *minimax* ne reflète donc pas vraiment la situation exacte de la partie, mais *en revanche*, nous pouvons dire que cette valeur est une bonne estimation de l'issue vers laquelle se dirige la partie. En fait, cette estimation est d'autant plus meilleure que le sous-arbre issu du nœud que l'on observe est complet, étoffé, c'est à dire que beaucoup de parties issues de cette séquence de coups ont été jouées. Car, plus de nouvelles parties se greffent sur l'arbre, plus ce dernier se rapproche de l'arbre complet de toutes les parties possibles à OTHELLO. Bien évidemment, le but n'est pas d'atteindre ce stade ultime, et cela serait d'ailleurs bien impossible (cf. I.2.2/ L'arbre de recherche). Un *minimax* appliqué à cet arbre complet donnerai pour chaque nœud l'estimation exacte de la situation. *Donc, plus notre arbre s'étoffera et se complétera, (et puisque les parties stockées dans l'arbre concernent des joueurs de haut niveau),* ***plus la valeur*** **minimax** ***se rapprochera de la valeur réelle.*** Mais il faut alors que OTAGE joue beaucoup plus de parties contre de forts programmes, car 300 parties représentent un acquis bien maigre : c'est un problème dont nous reparlerons dans la partie III.6.6. Nous supposerons pour la suite que la base de données de OTAGE est suffisamment fournie et nous allons voir comment nous utilisons notre arbre des parties déjà jouées pour ouvrir.

Utilisation de l'arbre : les ouvertures

Nous sommes dans la première phase d'une partie d'OTHELLO : les ouvertures. Supposons donc qu'une séquence de coups d'ouverture ont déjà été joués, et c'est maintenant à OTAGE de répondre. La séquence de coups nous amène à un certain nœud dans l'arbre. Deux cas se présentent :

1. tous les coups jouables à partir de cette position sont présents dans l'arbre : ce sont les fils du nœud étudié. OTAGE a donc déjà joué au moins une partie pour chacune des variantes possibles de cette ouverture. Il suffit donc pour OTAGE de répondre le coup dont la valeur *minimax* lui est la plus favorable (le MAX s'il est Noir, le MIN sinon).

2. OTAGE n'a pas joué au moins une fois toutes les variantes possibles de cette ouverture. Le nœud étudié a plusieurs fils, mais n'a pas tous les fils possibles (c'était le cas de la figure III-20). Voici comment OTAGE procède...
   Supposons que OTAGE joue les Noirs (resp. les Blancs).

   - si parmi les fils connus dans l'arbre, certains ont une valeur positive (resp. négative), cela signifie une partie gagnée et il faut jouer le coup correspondant à la plus grande valeur positive (resp. plus faible valeur négative), afin de gagner avec le plus de pions d'écart.

- si malheureusement aucun des fils connus ne possède de valeur positive (resp. négative), c'est que toutes les variantes que nous avons déjà jouées nous ont menés vers la défaite. Il faut donc essayer de jouer un des autres coups possibles non présents dans l'arbre. Si ces coups ne sont pas présents dans l'arbre, c'est parce qu'ils ont une valuation moins bonne au niveau de l'$\alpha\beta$, et c'est pour cela qu'ils n'avaient pas été joués. Il faut les essayer tout de même car il se peut qu'ils se révèlent intéressants dans la suite de la partie, bien qu'ils aient l'air moins bons a priori. Et puis de toute façon, nous sommes quasi certains de perdre si l'on joue un des coups de l'arbre malgré la meilleure valeur que leur attribue l'$\alpha\beta$.

  Nous sortons donc de la phase d'ouverture, et nous appelons l'$\alpha\beta$ avec des *coups interdits*. Nous décrirons exactement en quoi cela consiste dans la partie suivante III.6.3 / $\alpha\beta$ **avec** *coups interdits*.

En résumé Quand OTAGE doit jouer, il choisit une des réponses de sa "bibliothèque d'ouvertures" (i.e. l'arbre) s'il y en a une favorable. Sinon, s'il peut essayer un ou plusieurs coups encore jamais joués, il choisira (par le truchement de l'$\alpha\beta$) le meilleur de ceux-ci. Enfin, si aucun de ces deux cas ne se réalise, c'est que tous les coups jouables lui sont néfastes et il choisira alors de jouer la moins mauvaise des réponses.

$\alpha\beta$ **avec** *coups interdits*

Lorsque l'on appelle l'$\alpha\beta$, il se peut qu'on lui spécifie des *coups interdits*. Ceux-ci ne sont autres qu'une liste de coups à ne pas jouer. Ainsi, pour les ouvertures, lorsque l'on s'aperçoit que parmi les réponses que l'on connaît, toutes sont mauvaises, on essaie alors de jouer un coup encore jamais essayé : on appelle alors l'$\alpha\beta$ de la manière habituelle mais en lui précisant qu'il ne devra pas choisir un des coups que l'on connaît déjà car nous les savons "mauvais". L'$\alpha\beta$ va donc choisir le meilleur coup à jouer parmi ceux restants.

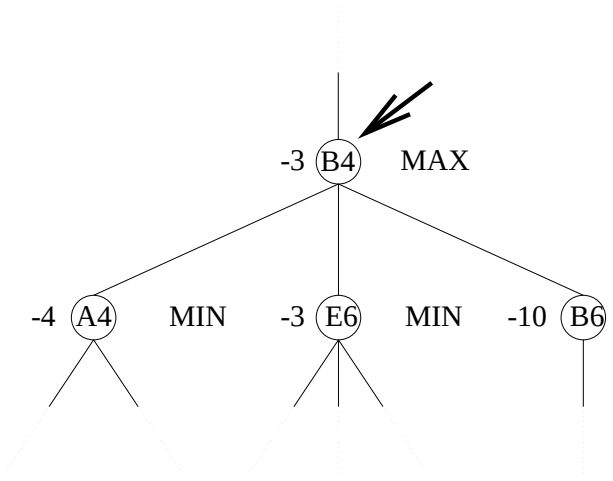


Figure III-22 : *ouvertures* : un exemple d'utilisation des *coups interdits*

Sur l'exemple de la figure III-22, c'est aux Noirs (OTAGE) de jouer, après le coup B4. Or les trois réponses qu'il connaît lui sont toutes défavorables et il peut au mieux espérer perdre 3 pions. Mais à ce stade des ouvertures (séquence C4 E3 F3 C5 D3 B4), OTAGE peut répondre l'un des coups suivants : C6, **A4**, D6, **E6**, B5 ou **B6**. OTAGE va donc appeler

l'$\alpha\beta$ avec les *coups interdits* suivants : A4, E6 et B6. L'$\alpha\beta$ va donc sélectionner le meilleur coups parmi ceux restants, c'est à dire : C6, D6, et B5.

III.6.4 Avantage des *coups interdits*

L'implantation des *coups interdits* permet non seulement d'essayer une nouvelle réponse lorsque l'on n'en connaît pas de favorable, mais cela permet aussi d'étoffer notre arbre des parties jouées. OTAGE essaye une réponse nouvelle, et par conséquence, une nouvelle branche se greffe à l'arbre : nous disposons alors d'une nouvelle variante.

Par ce mécanisme, si une ouverture se présente souvent, OTAGE explorera donc au fil des parties toutes les réponses possibles jusqu'à ce qu'il en trouve une qui lui soit favorable. Si aucune ne l'est, OTAGE aura donc exploré toutes les variantes possibles et sera donc en mesure la prochaine fois que cette ouverture se présentera de choisir, en l'absence de mieux, la moins mauvaise des réponses.

La technique des *coups interdits* permet donc au cours de la période d'apprentissage de OTAGE , d'étoffer l'arbre, et plus particulièrement le début, le haut de l'arbre (issu du premier coup C4), puisque les premiers coups d'une partie ne sont pas nombreux. OTAGE parcourt ainsi le plus de variantes possibles, et augmente sa bibliothèque d'ouvertures.

III.6.5 Un mécanisme non limité aux ouvertures

Le mécanisme que nous avons décrit ci-dessus ne se restreint pas seulement aux ouvertures.

En fait, pour déterminer quel coup répondre dans une position donnée, le programme recherche s'il n'y a pas une suite de coup dans l'arbre des parties déjà jouées qui l'amène à cette même position. Si tel est le cas, alors OTAGE répond en suivant le processus décrit plus haut.

Mais alors, il n'est pas nécessaire lorsqu'on sort des ouvertures connues par OTAGE, de déconnecter ce système. Car il arrive que l'on commence à jouer une certaine ouverture, puis qu'ensuite nous sortions de cette ouverture à cause d'un coup inattendu de l'adversaire, mais que deux ou trois coups plus loin la partie "rattrape" une suite de coups connue.

Comme de toute façon la détection d'une situation connue dans notre arbre ne se fait pas sur la séquence des coups depuis le premier coup C4 mais en fonction de la position actuelle du plateau de jeu, nous laissons toujours actif le système de réponse à l'aide de notre arbre. Ainsi, si au cours de la partie (et donc pas forcément pendant les ouvertures), OTAGE rencontre une position qu'il avait déjà observée durant une précédente partie, il répondra à l'aide du processus décrit ci-dessus. Il se peut donc qu'en cours de partie, OTAGE réponde donc instantanément et se comporte comme s'il connaissait la partie qui est en train d'être jouée (Ce qui est bel et bien le cas, en fait...).

III.6.6 Les difficultés rencontrées

L'apprentissage

Dans une première période d'apprentissage, la base de données de OTAGE qui ne démarre qu'avec 300 parties n'est pas assez importante. Il va donc falloir faire jouer à OTAGE beaucoup de matches afin qu'il se perfectionne. Remarquons tout de même que OTAGE apprend toujours constamment, chaque fois qu'il joue une partie. Il n'y a donc pas à proprement parler de période d'apprentissage et une autre période où il n'apprend plus. Par "période d'apprentissage", nous entendons une phase de développement où il va falloir

d'abord faire jouer OTAGE intensivement, lui faire jouer beaucoup de matches, afin d'étoffer sa base de données.

OTAGE a donc besoin de faire beaucoup de matchs ; d'un autre côté, plus l'adversaire est fort, meilleur est l'apprentissage. Le mieux serait de faire progresser OTAGE en le faisant jouer sur le serveur I.O.S. et en ne s'intéressant qu'aux très forts adversaires. Mais nous nous confrontons alors à un problème que nous avons déjà soulevé : les parties sur le serveur I.O.S. durent très longtemps puisque ce sont des matchs de "compétition" et l'on y joue avec les règles officielles c'est à dire 30 minute par joueur soit au maximum 1 heure par partie ! Si l'on y rajoute les déconnexions du serveur, la présence non constante des forts programmes, il devient difficile de faire jouer à OTAGE beaucoup de parties contre les forts adversaires.

Nous avons donc opté pour une solution consistant à faire jouer OTAGE contre lui-même, du moins dans un premier temps. Pourquoi ce choix et dans quelles conditions ? Tout d'abord, OTAGE est déjà un programme de très bon niveau. Donc le faire jouer contre lui-même lui permet de rencontrer un bon adversaire. Bien sûr, nous n'allons pas faire jouer OTAGE contre lui-même dans des parties de une heure : le problème resterait alors le même. Il s'affrontera dans des matches où chaque joueur aura 3 minutes au lieu de 30 pour jouer toute la partie. Certes, l'analyse produite au cours d'une partie de 3 minutes est moins bonne que celle d'une partie de 30 minutes : par manque de temps, les profondeurs atteintes durant la réflexion sont moins grandes. Donc, ces parties que nous stockerons ensuite dans l'arbre ne seront pas vraiment d'aussi bonne qualité que celles que nous pourrions obtenir sur le serveur I.O.S.. Cependant, nous cherchons juste, *dans un premier temps*, à étoffer l'arbre des parties jouées afin que OTAGE ait suffisamment d'expérience dans les ouvertures. Nous aurons tout le loisir, *par la suite*, de perfectionner OTAGE en le laissant combattre ses adversaires sur le serveur I.O.S.

L'autoplay

Afin de réaliser un autoplay permettant à OTAGE de jouer contre lui-même, nous avions deux possibilités :

1. modifier le source de OTAGE afin qu'il puisse jouer les deux adversaires
2. faire en sorte que le processus se duplique, se dédouble, l'un jouant les Blancs, l'autre les Noirs.

Nous avons opté pour la solution la plus facile. OTAGE a été programmé à son origine dans l'optique où il n'aurait qu'une couleur à jouer. Si l'on choisit la première solution, il va falloir modifier tout le code source du programme. En revanche, si l'on se contente de dupliquer le processus, il sera juste nécessaire de passer en paramètre à chacun des processus la couleur qu'il joue et d'établir une connexion par des "pipes" afin qu'ils communiquent ensembles (lecture et écriture des coups joués). C'est donc cette option que nous avons choisie. Le programme se déroule alors comme suit :

Déroulement du programme	
Début d'initialisation	
Saisie des paramètres de la ligne de commande	
Reconnaissance du mode autoplay	
Connexion père-fils	
Duplication du programme en un père et son fils	
père	fils
joue les Noirs	joue les Blancs
établissement des pipes	établissement des pipes
Suite et fin de l'initialisation	Suite et fin de l'initialisation
Jouer la partie	Jouer la partie
Fin de partie : le père reste en vie	Fin de la partie : fin du processus
Traitement d'après match Mise à jour : -des paramètres -des bases de données	. . . .

Cette solution ne comporte pas d'inconvénient au niveau du temps CPU lors de l'exécution car chaque joueur joue l'un après l'autre. L'occupation CPU n'est pas doublée : le père reste en veille quand c'est au fils de jouer et vice versa. En revanche, l'espace mémoire occupé est plus grand car certaines données sont dupliquées.

Apparition d'un biais

Première déviance OTAGE jouant contre lui-même, l'arbre va se développer en fonction des évaluations faites au cours des parties par OTAGE **uniquement**. C'est un problème dont nous avons déjà quelque peu parlé : nous n'avons pour juger que la fonction d'évaluation d'OTAGE , et nous ne bénéficions plus de "l'avis" des autres programmes. Ceci n'est pas grave à ce stade car nous désirons juste ans un premier temps développer suffisamment l'arbre des ouvertures pour que OTAGE joue encore mieux. Il pourra ensuite corriger ses déviances et s'affiner en rencontrant les autres programmes.

Deuxième déviance Nous l'avons détectée en faisant jouer comme prévu OTAGE contre lui-même. Le principe de développement de notre arbre d'ouvertures est bon : si lorsque l'on doit répondre nous ne disposons pas dans l'arbre de coup favorable et que tous les coups n'ont pas été essayés, alors testons le meilleur de ces coups restants et cela rajoutera une branche à l'arbre. Notre but en faisant jouer OTAGE contre lui-même est de développer *les ouvertures*, c'est à dire surtout les parties hautes de l'arbre, proches de la racine, qui sont les premiers coups des parties. Nous désirons que OTAGE en sache le plus possible sur ces premiers coups d'une partie afin que OTAGE dispose d'une grande panoplie d'ouvertures déjà expérimentées, qu'il ait reconnu les meilleures, et jouées les diverses variantes, afin qu'il soit à même de bien ouvrir et si possible le plus longtemps possible pour gagner du temps.

Malheureusement, nous constatons que OTAGE a plutôt tendance à ne développer qu'une partie de l'arbre. Il étoffe les variantes et sous-variantes de cette partie de l'arbre, ce qui fait qu'il est très expérimenté sur cette partie des ouvertures, mais très peu sur le reste. Si jamais un des adversaires sur I.O.S. ouvre en dehors des ouvertures que connaît très bien OTAGE , ce dernier sera autant perdu qu'avant. Cette déviance s'explique...

Nous allons envisager une situation. Imaginons que OTAGE joue contre lui-même. Les Noirs ouvrent C4. Suite à ce coup, 3 réponses sont possibles ; mais pour l'instant, OTAGE ne connaît que E3 (les autres sont C3 et C5). La valeur de cette réponse E3 est négative : c'est à dire favorable aux Blancs. Les Blancs vont donc répondre E3 : il y a eu pour l'instant la séquence C4 E3. Comme l'arbre a été évalué par *minimax*, si le coup E3 vaut moins quelque chose, cela signifie que quels que soient les coups appartenant à l'arbre que pourrait jouer le camp Noir, Blanc disposera dans l'arbre des réponses nécessaires pour gagner la partie. Donc, dès que Noir aura la possibilité de tester un coup qui n'est pas dans l'arbre des ouvertures, il le fera, rajoutant ainsi une nouvelle valeur, une nouvelle branche. Deux cas se présentent alors à l'issue du match :

1. Noir gagne la partie. La nouvelle branche ainsi ajoutée introduira une feuille de valeur positive (favorable aux Noirs) et par le jeu du *minimax*, le deuxième coup E3 deviendra positif. Noir aura trouvé le moyen de contrer la réponse E3 des Blancs. Du coup, à la partie suivante, les Blancs ne joueront plus E3 suite à C4, mais essaieront une des deux autres réponses possibles : C3 ou C5. L'arbre s'étoffe ainsi et tout va bien.

2. Noir perd encore la partie. Malheureusement, c'est ce cas-ci qui a le plus de chance d'arriver et ceci pour deux raisons :

   (a) lorsque Noir essaye un coup différent de ceux déjà testés à un endroit quelconque de l'arbre, ce nouveau coup a encore moins de chance que les autres de le mener vers la victoire. En effet, à un certain niveau de l'arbre, on essaie de toute façon les supposées (estimation faite par l'$\alpha\beta$) meilleures réponses d'abord. Donc ce nouveau coup que l'on tente est normalement encore moins bon. (nous rappelons que l'on tente tout de même ces coups, pour étoffer l'arbre, et parce que l'estimation faite par $\alpha\beta$ n'est justement qu'une estimation, et un coup jugé mauvais peut parfois se révéler bon par la suite).

   (b) Deuxièmement (remarque valable bien sûr que pour les Noirs), il est connu que les Noirs ont plus de chances de perdre que les Blancs au cours d'une partie d'OTHELLO. Ceci est dû au fait que ce sont les Blancs qui jouent en dernier, et qui donc retournent en dernier des pions. Sur un plateau d'OTHELLO où tous les pions sont posés sauf un, lorsque Blanc pose son dernier pion, il y a des chances pour qu'il retourne un grand nombre de pions et fasse la décision.

Les Noirs étant défavorisés, trouver un nouveau coup durant la partie permettant aux Noirs de renverser la valeur du coup E3 en un nombre positif n'est pas évident. Cela n'est pas introuvable aussi, car le coup E3 n'est pas non plus connu pour être gagnant. Mais la fréquence de tels coups est rare et le cas 1 se présentera plus rarement. Si on laisse OTAGE jouer ainsi contre lui-même, le camp Noir essaiera absolument de trouver un coup qui lui est favorable, et OTAGE va du coup tester de partie en partie toutes les variantes issues de E3. Résultat : c'est le sous-arbre issu de E3 qui se développe mais pas les autres parties de l'arbre. Lors d'un match contre un adversaire autre que OTAGE , si OTAGE joue C4 et que l'adversaire répond autre chose que E3, OTAGE ne sera pas expérimenté sur cette ouverture. Il y a un développement asymétrique de l'arbre.

Bien sûr, cet exemple n'est pas la seule situation pouvant amener un développement non uniforme de l'arbre. Nous insistons d'ailleurs encore sur le fait que le but général recherché n'est pas non plus d'obtenir un arbre uniforme : *une fois l'arbre déjà suffisamment étoffé*, il est naturel que se développe *ensuite* la partie de l'arbre correspondant aux meilleures branches, de sorte que OTAGE explore un maximum de variantes des ouvertures

les plus intéressantes. Seulement nous, dans un premier temps, nous cherchons juste à ce que OTAGE augmente sa bibliothèques d'ouvertures, afin qu'il inspecte un peu toutes les ouvertures possibles et en déduise lui-même les plus intéressantes : il pourra ensuite affiner ses ouvertures et développer les parties de l'arbre qui sont intéressantes en jouant contre ses adversaires. Comment pour l'instant résoudre le problème qui nous préoccupe ?

La solution apportée

Notre problème est donc d'obtenir une première bibliothèque d'ouvertures complète pour permettre déjà à OTAGE de bien se débrouiller dans la première phase d'une partie. Pour se faire, le seul moyen est en fait de forcer OTAGE à explorer toutes les ouvertures possibles, puisque le système d'amélioration de la base de données qui est en place ne sert qu'au perfectionnement des ouvertures et non à la constitution d'une base de départ.

Nous allons donc faire jouer par OTAGE (contre lui-même) tous les débuts de parties possibles, en se limitant à une certaine profondeur. Nous avons choisi des séquences d'une longueur de 6 coups en nous fondant sur deux critères : le nombre de séquences explose exponentiellement avec la longueur de celles-ci, mais à l'inverse plus les séquences seront longues, plus OTAGE possédera une vue experte sur les ouvertures. Nous avons donc généré toutes les ouvertures possibles de 6 coups (le premier étant toujours C4) et nous forcerons les 6 premiers coups des parties de OTAGE contre lui-même. Il ne reste plus maintenant qu'à laisser OTAGE jouer toutes ces parties. Mais déjà, rien qu'avec 6 coups, nous avons 2050 ouvertures possibles. Si l'on ne veut pas se contenter seulement d'une seule partie jouée pour chacune de ces ouvertures, cela va nous demander énormément de temps. Un seul passage sur toutes ces ouvertures représente déjà 205 heures soit 8 jours et demi (avec seulement 3 minutes par partie). Comme il serait vraiment préférable d'effectuer beaucoup plus qu'un seul passage, il convient alors de répartir OTAGE sur plusieurs machines afin de diviser par autant le temps nécessaire à l'apprentissage de OTAGE .

Chapitre IV

Quelques résultats

IV.1 Amélioration de l'$\alpha\beta$

Afin d'observer ces améliorations, nous allons jouer une partie quelconque contre OTAGE mais sur deux versions différentes. L'une est la version de base, l'autre une version comportant les améliorations apportées à l'$\alpha\beta$. Cette dernière possède la technique des variantes. En revanche, la méthode d'ouverture est la même, par fichier d'ouvertures, afin que les parties jouées soient les mêmes. Les versions n'utilisant plus le fichier d'ouvertures mais l'arbre des ouvertures commencent les parties différemment. Enfin, l'implantation du nouvel ordre des coups n'y figure pas, toujours pour la même raison, afin que les coups répondus soient encore les mêmes en début de partie pour les deux versions.

OTAGE joue en premier et a donc le camp Noir. Le temps de la partie sera le temps officiel, le temps de compétition, c'est à dire 30 minutes maximum pour que OTAGE joue tous ses coups. Le déroulement de la partie et toutes les données correspondantes sont regroupées dans le tableau suivant dont nous expliquons les en-têtes :

- Coup joué : coup répondu par OTAGE
- Nb(amélioré)-Nb(base) : c'est le nombre de feuilles explorées par la version avec $\alpha\beta$ amélioré pour trouver ce coup moins le nombre de feuilles explorées par la version de base pour trouver ce même coup.
- Tps(amélioré)-Tps(base) : c'est la différence de temps utilisé depuis le début de partie par la version améliorée moins le temps dépensé depuis le début de la partie par la version de base.
- Humain : le coup que nous répondons à OTAGE.

No	Coup joué	Nb(amélioré)-Nb(base)	Tps(amélioré)-Tps(base)	Humain
1	C4	0	0	E3
2	F4	0	0	G3
3	E6	-164 745	+3"28	D6
4	F5	-104 620	-0"48	F6
5	G6	-169 711	-9"30	G4
6	G5	+144 274(-191 649)	+15"97(-41"22)	H6
7	F3	-200 589	+6"33	F2
8	H4	-807 834	-38"17	H5
9	H7	-1 359 011	-1'49"24	C5
10	B5	-308 951	-2'02"86	C6
11	H3	-308 048	-2'17"57	A5
12	H2	-149 289	-2'23"75	D3
13	F1	+331 873(-62 046)	-1'36"05(-2'50"87)	G1
14	H1	-51 001	-1'30"36	G2
15	E2	-27 018	-1'22"56	E1
16	F7	-153 151	-1'11"84	G8
17	D1	-68 404	-1'08"60	G7

Remarque Les parties divergent ensuite : la version de base joue C2 alors que la version améliorée joue C7. La cause de cette divergence est l'utilisation par la version améliorée des *patterns* sensibles à la couleur : cela entraîne des évaluations (légèrement) différentes. A ce stade de la partie, deux bons coups ont une valuation très proche (C2 et C7) et les différences entre les deux versions d'OTAGE amènent la version de base à évaluer C2 à hauteur de 1120 (et C7 un petit peu moins), alors que la version améliorée attribue une valeur de 1118 à C7 (et un tout petit peu moins pour C2).

Lignes 6 et 13 Deux lignes de ce tableau comportent des chiffres entre parenthèses... Ces deux lignes traduisent deux occasions où la version améliorée a étudié le coup plus profondément. Après la ligne 5 du tableau, la version améliorée a déjà économisé 9 secondes et 30 centièmes par rapport à la version de base. Pour le coup suivant, c'est à dire G5, la version de base va jusqu'à la profondeur 9 et s'arrête. Lorsque la version améliorée termine son analyse à la profondeur 9, elle a comparativement économisé 41 secondes et 22 centièmes (chiffre entre parenthèses). Vu le peu de temps qu'elle a pour l'instant dépensé, cette version estime alors qu'elle peut se lancer dans une recherche à la profondeur 10 suivante. Nous voyons donc ici tout l'intérêt des améliorations que nous avons apportées. Avec le temps gagné dû à un plus grand élagage, la version améliorée peut se permettre à plusieurs reprises d'explorer plus profondément et donc de jouer mieux.

La ligne No 6 peut donc se lire ainsi : la version améliorée semble avoir parcouru plus de feuilles et pris plus de temps par rapport à la version de base, pour jouer le même coup ; mais en fait, la version améliorée a produit une analyse plus profonde et meilleure, grâce au temps gagné. Il aurait très bien pu se produire qu'à l'aide de l'analyse en profondeur 10, OTAGE s'aperçoive qu'il y a un meilleur coup à répondre que G5. De plus, les *variantes* ainsi produites sont augmentées d'un coup comparativement à celles de la version de base.

Analyse des résultats Il ressort de ce tableau, que la version améliorée analyse beaucoup moins de feuilles que la version de base. Cela a pour conséquence de permettre à OTAGE

de jouer plus vite, de dépenser moins de temps. Ce temps économisé pourra être mis à profit pour analyser plus profondément certaines positions ou pour l'analyse exhaustive de fin de partie. Si l'on fait le total des différences constatées dans le tableau précédent, nous obtenons les résultats suivants pour seulement 17 coups joués par OTAGE :

- si l'on se place à profondeur égale : l3 818 019 feuilles étudiées en plus par la version de base, pour un même résultat.
- si l'on observe juste la différence de feuilles effective : 3 088 177 feuilles explorées en plus, pour le même résultat.
- Après 17 coups, la version de base a effectivement utilisé 1'08"60 de plus pour jouer pourtant les mêmes coups
- Et à profondeur d'étude égale, la version de base utilise 3'20"61 de plus pour jouer les mêmes coups.

En revanche, si l'on observe les coups où les deux versions sont allées à la même profondeur, la version améliorée ne gagne pas toujours du temps sur la version de base, puisque par exemple, l'écart entre les deux versions diminue entre les coups No 14 et No 17. Pourtant, pour ces coups-ci, OTAGE amélioré explore moins de feuilles ! Mais il faut se rappeler que l'utilisation des variantes entraîne un coût en temps relatif à la génération de ces variantes. Pour les coups No 14 à No 17, le temps gagné grâce à l'exploration de moins de feuilles ne compense pas le temps perdu à générer les variantes. Néanmoins, il est très important de noter que si parfois il arrive à la version améliorée de perdre du temps par rapport à la version de base, cette perte n'est jamais énorme ; en revanche, lorsque la version améliorée gagne du temps, elle en gagne beaucoup ! Le bilan sur une partie est donc largement positif. D'ailleurs, malgré les pertes des coups 14, 15, 16 et 17, la version améliorée dispose encore de 1'08"60 d'avance (voire de 3'20"61 si l'on compare à profondeur égale) sur la version de base.

Nullus homo est Nous avons perdu les deux parties du tableau ci-dessus (celle se continuant par C2 et l'autre par C7) sur le même score honteux de 64 à 0...

IV.2 Un coup d'oeil sur les variantes

Nous allons jeter un oeil sur les variantes produites au cours de la recherche d'un des coups du début de la partie ci-dessus. Plaçons-nous au moment où on a déjà la séquence suivante : C4 E3 F4 G3 E6 D6 F5 F6. C'est donc à OTAGE de jouer maintenant et il a le choix entre 11 coups : E2 F2 G4 G5 G6 C7 D7 E7 F7 H2 G7.

La première analyse à la profondeur 5 produit les 11 variantes suivantes où P désigne la variante principale :

P	G5 H4 H5 G4 H3	1	F2 C5 C6 B3 D7	2	G4 F3 G5 C3 D3	3	E2 C5 G4 D3 C3
4	G6 C3	5	C7 C6	6	D7 C3	7	E7 C5
8	F7 C3	9	H2 C3	10	G7 D3		

La meilleure séquence est donc G5 H4 H5 G4 H3 ce qui signifie que OTAGE compte jouer G5. L'adversaire étant supposé fort, on estime qu'il jouera tout le temps le meilleur

coup à sa disposition. Il devrait donc répondre H4, OTAGE suivra alors par H5, puis ainsi de suite, G4 puis H3.

Les autres coups possibles pour OTAGE sont moins bons comme nous le montrent les autres variantes qui ne sont autres, par définition, que les coups de réfutation. Si OTAGE joue E7, la seule réponse C5 de son adversaire amènera OTAGE dans une position beaucoup moins bonne que si OTAGE avait joué G5[1] . Les coups associés aux variantes 4, 5, 6, 7, 8, 9 et 10 sont donc mauvais puisqu'une seule réponse de l'adversaire entraîne une position de valeur nettement moins bonne que si OTAGE répond par G5. Cependant, nous pouvons remarquer que les coups F2, G4 et E2 associés aux variantes 1, 2 et 3 sont plutôt des bons coups, a priori, puisqu'il faut toute une séquence de 5 coups pour que l'adversaire amène la position dans une situation plus défavorable que celle induite par le coup G5.

OTAGE passe maintenant à une analyse de profondeur 6 (en utilisant les variantes ci-dessus). Les variantes produites par cette analyse sont maintenant :

P	G5 C3 G4 B4 D3 C5	1	F2 C5	2	G4 F3	3	E2 C5
4	G6 C3	5	C7 C6	6	D7 C3	7	E7 C5
8	F7 C3	9	H2 C3	10	G7 D3		

La réponse G5 s'affirme ici comme le meilleur coup à jouer. La meilleure branche est devenue : G5 C3 G4 B4 D3 C5 et nous nous apercevons que cette suite de meilleurs coups engendre une situation qui est largement meilleure que celles que pourraient engendrer n'importe quel autre coup de OTAGE autre que G5. Même les coups F2, G4 et E2 qui nous avaient apparus bons sont réfutés par un seul coup de l'adversaire. Nous voyons ici l'importance d'analyser le plus profondément possible : on a une bien meilleure vue de la situation (comme nous le verrons encore pour la profondeur 8).

L'analyse à la profondeur 7 conforte encore le coup G5 :

P	G5 H6 G4 C5 C7 C6 C3	1	F2 C5	2	G4 F3	3	E2 C5
4	G6 C3	5	C7 C6	6	D7 C3	7	E7 C5
8	F7 C3	9	H2 C3	10	G7 D3		

La profondeur 8 est très instructive car elle chamboule la situation :

P	G6 C3 E7 C5 F7 G5 F3 F8	1	F2 C5	2	G4 F3	3	E2 C5
4	G5 H6 F3 C3 E2 G4 C5 F2	5	C7 C6	6	D7 C3	7	E7 C5
8	F7 C3	9	H2 C3	10	G7 D3		

Oui ! La variante principale a changé ! Le meilleur coup à jouer n'est plus maintenant G5 mais G6 ! Seule une analyse à la profondeur 8 a pu nous montrer cela. Alors que le coup G6 apparaissait jusqu'à maintenant comme inintéressant (il suffisait à l'adversaire de répondre C3 pour que le coup G6 soit moins bon), nous nous apercevons que c'est à la profondeur 8 que la meilleure séquence issue de G5 devient en fait moins bonne que la meilleure séquence issue de G6. Nous pouvons analyser ce fait comme une dégradation de la position engendrée par le coup G5, dégradation que nous ne constatons que 8 coups plus tard ; de sorte que la situation engendrée par le coup G6 devient finalement meilleure. D'ailleurs, à cette profondeur, les deux coups sont assez proches, puisqu'il faut quand même une séquence de 8 coups pour les départager.

La dernière analyse à la profondeur 9 consacre le coup G6 :

[1] c.à.d. quels que soient les coups joués après C5, la position sera de toute façon moins avantageuse

P	G6 H6 F3 C3 E2 B4 E7 F8 D3	1	F2 C5	2	G4 F3	3	E2 C5
4	G5 H6	5	C7 C6	6	D7 C3	7	E7 C5
8	F7 C3	9	H2 C3	10	G7 D3		

Cette fois, la profondeur d'analyse supplémentaire nous montre que G6 est bien le meilleur coup ; la valeur de la position générée par la suite des 9 coups issus de G6 est la plus forte. Si l'on joue une quelconque des autres réponses possibles, un seul coup de l'adversaire permet de réfuter cette réponse et nous amène dans une position moins bonne que celle obtenue par G6. Nous remarquons que même la réponse G5 est réfutée par un seul coup (H6).

Comme nous le voyons, il est vital de pouvoir analyser une position le plus profondément possible. Si nous n'étions pas allé à la profondeur 8 (puis 9), OTAGE aurait joué G5 au lieu de G6, et se serait exposé à voir l'adversaire, s'il joue bien, répondre H6 et amener le jeu dans une position moins bonne que ce qu'elle aurait pu être. C'est donc tout à l'avantage d'OTAGE d'avoir maintenant des techniques lui permettant d'analyser les positions plus vite et en moins d'effort : il peut ainsi employer le temps gagné à analyser plus profondément, et du coup, OTAGE joue beaucoup mieux.

IV.3 Changement de l'ordre des coups

Nous constatons que le nouvel ordre des coups que nous avons mis en place entraîne bel et bien de nouveaux élagages : le nombre de feuilles parcourues est encore moins grand. Bien sûr, les coupures induites se font dans une proportion beaucoup moins grande que celles occasionnés par les techniques d'optimisation implantées au niveau de l'$\alpha\beta$. Cela représente tout de même un gain supplémentaire dont il serait regrettable de se passer, d'autant qu'il ne coûte rien au niveau du temps. Cet ordre, tout comme l'ancien, est chargé en début d'exécution, à l'initialisation, et reste ensuite valable et actif pour toutes les parties à venir et ne génère aucun traitement supplémentaire.

Chapitre V

Conclusion

Nous avons amélioré plusieurs parties de OTAGE. Le programme parcourt maintenant d'une manière très optimisée l'arbre de recherche. Il joue ainsi plus vite et réalise des analyses plus profondes. Nous avons développé une approche des ouvertures différente et de meilleure qualité. Les performances de OTAGE devraient s'améliorer au fur et à mesure qu'il acquerra de l'expérience.

Au stade actuel, il reste à terminer l'apprentissage de la bibliothèque d'ouvertures et nous pensons déjà aux autres améliorations qu'il est encore possible d'apporter à OTAGE. Ses performances pourraient être accrues s'il pouvait utiliser le temps de réflexion de son adversaire pour analyser la position. Il serait aussi intéressant de paralléliser l'exploration de l'arbre de recherche.

Le programme OTAGE n'en est donc pas à sa version définitive, et nous pouvons espérer le voir progresser davantage...

Annexe A

Le jeu d'OTHELLO

A.1 Présentation

OTHELLO est un jeu à deux joueurs, qui s'affrontent sur un plateau de soixante quatre cases. Un joueur possède les pions blancs, l'autre les pions noirs. Le gagnant est celui qui, à la fin de la partie, détient le plus grand nombre de pions.

Les jeux de plateau à deux joueurs sont d'une manière générale un terrain propice pour la mise en œuvre des techniques de l'Intelligence Artificielle. Parmi tous les jeux de ce type, quelques uns se distinguent particulièrement et sont devenus des "classiques", pour le développement, la recherche et la mise en application des techniques de l'Intelligence Artificielle. Ce sont les échecs, les dames et notamment OTHELLO, très étudié pour ses spécificités que nous détaillerons plus loin : nombre de coups fini, pas de cycle dans les positions successives du plateau de jeu, etc.

A.2 Le jeu

A.2.1 Le plateau de jeu

Le plateau de jeu est constitué de soixante quatre cases, sans aucune distinction de couleur (la couleur des cases est généralement verte). Les pions sont des disques dont l'une des faces est blanche, l'autre noire. Un joueur représente le camp blanc, l'autre les pions noirs. Le joueur Noir posera ses pions face noire vers le haut, et vice versa pour les Blancs. La position de départ est montrée sur la figure A-1.

A.2.2 Les règles du jeu

Ce sont les Noirs qui jouent en premier. Ensuite, chaque joueur joue l'un après l'autre. La partie se termine si toutes les cases du plateau de jeu ont été remplies, ou bien si aucun des deux joueurs ne peut jouer. On compte alors le nombre de pions de chacun et l'on détermine le vainqueur.

Poser un pion

Pour pouvoir poser un pion sur une case, deux conditions doivent être remplies :

1. On ne peut poser un pion que sur une case vide.

UN PION

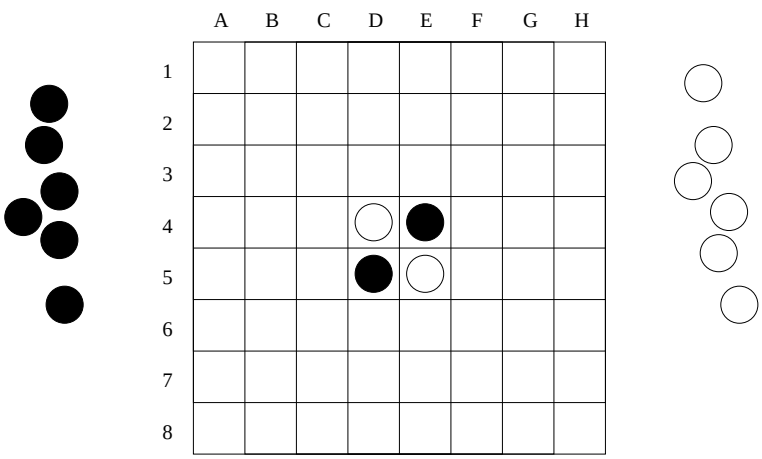


Figure A-1 : Le plateau de jeu : position de départ

2. On ne peut poser un pion sur une case que si cela permet au joueur qui pose ce pion de "retourner" des pions adverses.

Si aucune de ces deux conditions n'est remplie, le joueur doit passer son tour. Si le joueur ne peut poser un pion que sur une seule case et que cela le mette dans une mauvaise posture, il *doit* jouer quand même ce coup. *Il y a obligation de jouer. Il n'est pas permis de passer sans y être obligé.*

Retourner des pions adverses

Si le fait de poser un pion sur le plateau permet d'encadrer dans au moins une des huit directions possibles (cf. figure A-2), une suite continue de pions adverses, ceux-ci sont retournés, changent donc de couleur, et vous appartiennent maintenant.

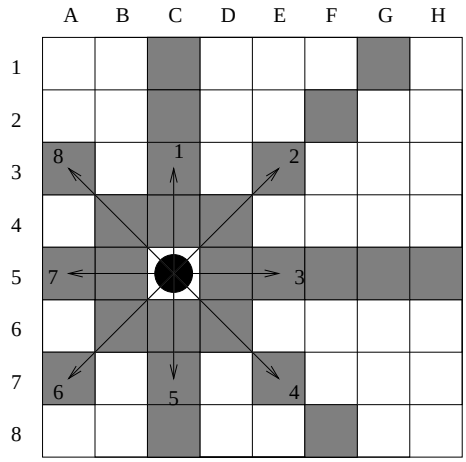


Figure A-2 : Les règles du jeu : les huit directions possibles

Ainsi, dans la position de départ, les Noirs ne peuvent jouer qu'aux seuls endroits marqués d'une croix sur la figure A-3. Si par exemple Noir décide de jouer en C4 (cf. figure A-4), le pion blanc en D4 est retourné et la position finale devient comme montré sur la figure A-5 ; ce sera aux Blancs de jouer ensuite.

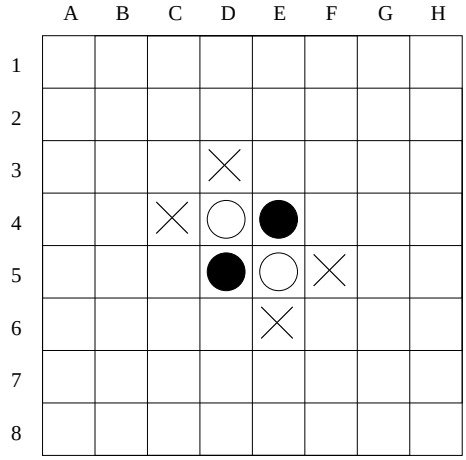


Figure A-3 : Retournement (1er exemple) : position de départ

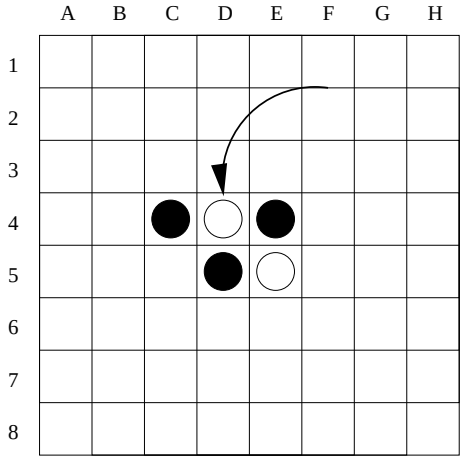


Figure A-4 : Retournement (1er exemple) : position intermédiaire

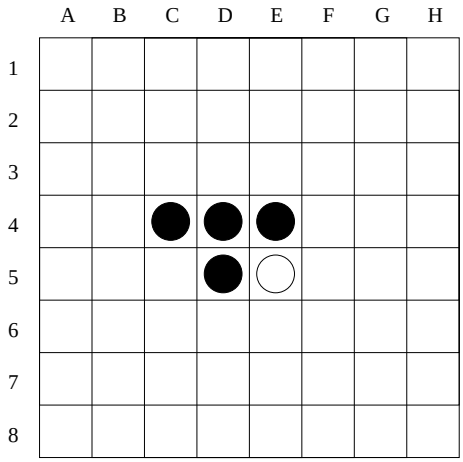


Figure A-5 : Retournement (1er exemple) : position finale

Un dernier exemple pour bien visualiser le retournement de pions : dans la position de la figure A-6, supposons que c'est aux Noirs de jouer. S'ils décident de jouer en E3, les pions blancs marqués d'une croix sur la figure A-7 vont être retournés. On obtiendra la position de la figure A-8.

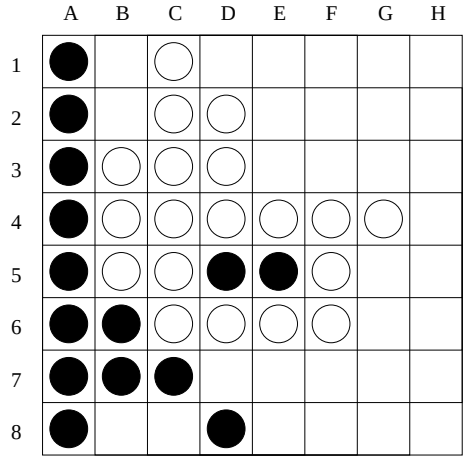


Figure A-6 : Retournement (2ième exemple) : position de départ

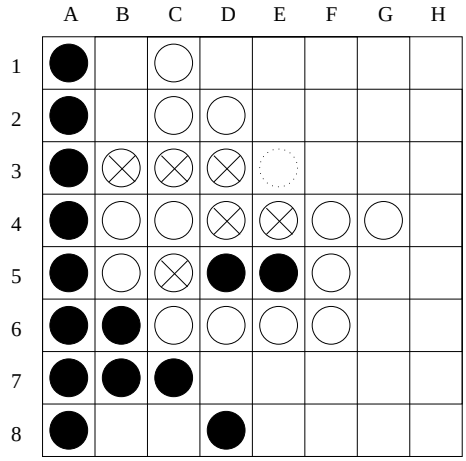


Figure A-7 : Retournement (2ième exemple) : position intermédiaire

Le temps imparti

Chaque joueur dispose de trente minutes pour jouer tous ses coups. S'il n'y parvient pas, il est déclaré perdant au temps. Cette durée de trente minutes constitue la durée officielle, mais bien sûr, comme aux échecs, les deux joueurs peuvent s'entendre pour faire une partie amicale sans contrainte de temps. De plus, il est à noter qu'il existe aussi des parties en Blitz[1] dont la durée peut varier de cinq à quinze minutes par joueur.

[1] des parties "éclair"

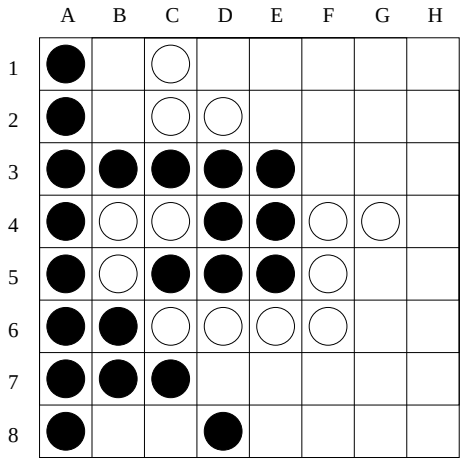


Figure A-8 : Retournement (2ième exemple) : position finale

A.3 Le serveur I.O.S.

I.O.S. est un serveur d'INTERNET sur lequel se retrouvent tous les grands amateurs du jeu d'OTHELLO, et des programmes comme OTAGE. L'intérêt de ce serveur est qu'il organise des rencontres entre tous les joueurs présents (humains ou machines) et nous allons pouvoir confronter notre programme à d'autres adversaires. Le classement se fait à l'aide de points calculés selon un barème similaire aux points ELO des Échecs. Deux types de matches sont proposés, en Blitz ou partie normale, et il y a donc deux classements différents.

Le serveur I.O.S. est d'une grande utilité de par la qualité des joueurs présents : tous les meilleurs programmes existants se retrouvent sur I.O.S., voire même parfois le programme champion du monde LOGISTELLO. Il est fondamental de permettre aux machines de s'affronter, car lors du développement d'un jeu d'OTHELLO, on ne dispose pas d'adversaire suffisamment fort (si ce n'est lui-même) pour évaluer le programme et pour lui faire acquérir de l'expérience.

Annexe B

Figures des résultats de divers algorithmes sur le même arbre

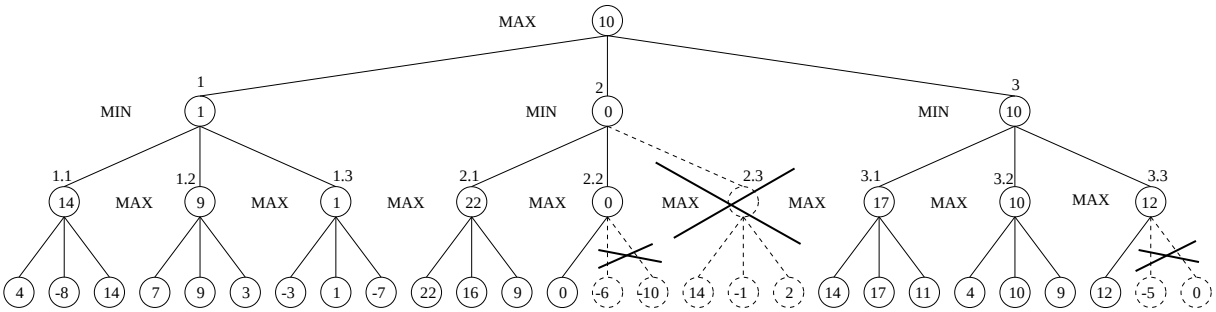


Figure B-1 : $\alpha\beta$

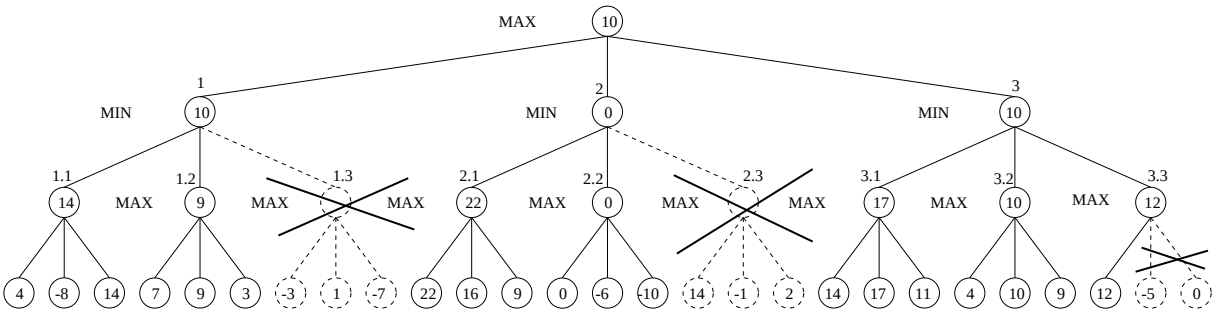


Figure B-2 : $\alpha\beta$ avec *meilleur coup*

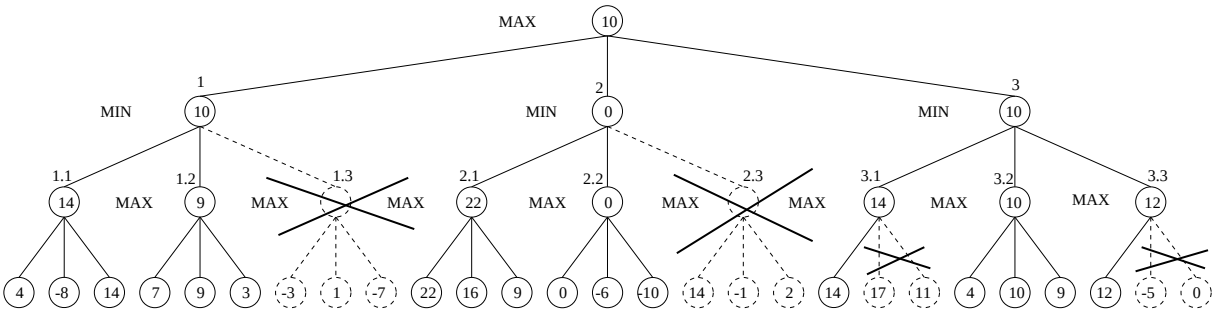


Figure B-3 : $\alpha\beta$ avec *meilleure branche*

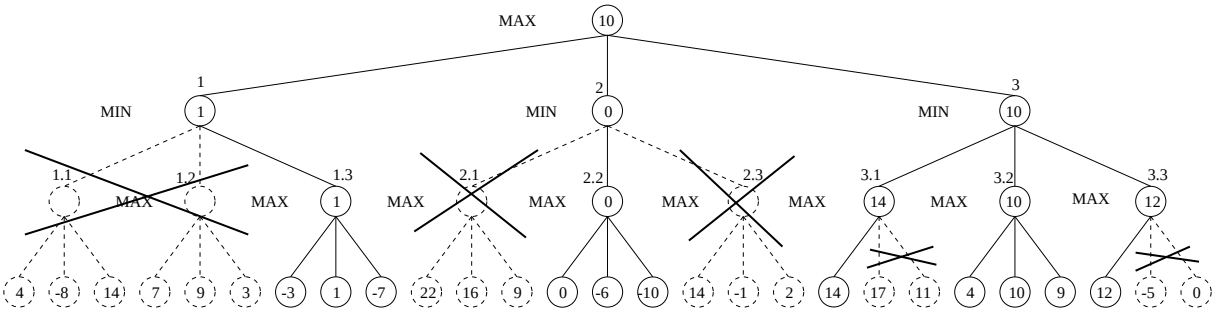


Figure B-4 : $\alpha\beta$ avec *tables de réfutation*

Annexe C

Structure modulaire du programme OTAGE

OTAGE est programmé en langage C et dispose de nombreux modules chargés chacun de réaliser des fonctionnalités regroupées autour d'un même sens. En voici l'énumération :

- `main.c` : constitue le corps principal de OTAGE.
  - saisie de la ligne de commande
  - initialisation
  - s'il y a lieu, connexion au serveur I.O.S.
  - s'il y a lieu, duplication du processus (mode autoplay)
  - lancement du jeu
- `alpha.c` : s'occupe de tout ce qui a trait à la recherche par $\alpha\beta$
  - procédure $\alpha\beta$
  - récupération des *variantes*
- `book.c` : s'occupe de tous ce qui concerne les ouvertures
  - initialisation de l'arbre des parties déjà jouées
  - recherche d'une réponse d'ouverture
  - *coups interdits*
  - rajout dans la base d'une nouvelle ouverture en fin de partie
- `client.c` : module d'interfaçage avec le serveur I.O.S.
- `endgame.c` : implante tout ce qui est relatif à la stratégie de fin de partie
- `eval2.c` : module s'occupant de la fonction d'évaluation mais aussi du mode replay et de la réévaluation des patterns en fin de partie. C'est ce module qui modifie les paramètres du programme après une partie.
- `osc.c` : module qui fait administre la partie. Il contient l'algorithme principal de jeu, et il s'adapte aux divers modes possibles.

- `search.c` : ce module gère tout ce qui concerne la recherche. C'est lui qui fera appel aux fonctions présentes dans le module `alpha.c`. Il implante la couche au-dessus de la simple recherche par $\alpha\beta$
- `valid.c` : module contenant seulement deux fonctions et servant à déterminer si un coup est valide ou non

Bibliographie

[Abr91] Abramson (Bruce). – The expected-outcome model of two-player games, 1991.

[Alt91] Althofer (I.). – Selective trees and majority systems: Two experiments with commercial chess computers., 1991.

[AN77] Akl (S.G.) et Newborn (M.M.). – The principle continuation and the killer heuristic. *In : ACM Annual Conference*, pp. 466–473.

[AS94] Alliot (J.-M.) et Schiex (T.). – Intelligence artificielle et informatique théorique, Mars 1994.

[AVAD52] Adelson-Velsky, Arlazorov et Donskoy. – *Algorithms for games.* – Springer-Verlag, 1952.

[AvdMvdH91] Allis (L.V.), van der Meulen (M.) et van den Herik (H.). – Alpha-beta conspiracy-number search, 1991.

[BS90] Billman (Dorrit) et Shaman (David). – Strategy knowledge and strategy change in skilled performance: A study of the game othello, Summer 1990.

[Bur] Buro (Micheal). – Probcut: A powerful selective extension of the alphabeta algorithm. – ftp://ftp.uni-paderborn.de/unix/othello/ps-files/.

[Bur94] Buro (Micheal). – *Techniques for the Evaluation of Game Positions Using Examples.* – Thèse de Doctorat, University of Paderborn, Germany, 1994.

[Bur95] Buro (Micheal). – Statistical feature combination for the evaluation of game postions. *Journal of AI Research*, 1995. – ftp://ftp.uni-paderborn.de/unix/othello/ps-files/combi.ps.gz.

[Chi88] Chi (Ping-Chung). – *In search of Better Decision-Making in Computer Game Playing.* – Thèse de Doctorat, University of Maryland College Park, 1988.

[CM83] Campbell (Murray) et Marsland (T.A.). – A comparison of minimax tree search algorithms, 1983.

[Fre80] Frey (P.W.). – Machine othello. *Personal Computing*, vol. 4, n° 7, 1980, pp. 89–90.

[Gup89] Gupton (Greg). – Genetic learning algorithm, applied to the game of othello. *In : Heuristic Programming in Artificial Intelligence, the first computer Olympiad*, éd. par Levy (David) et Beal (D.F.), pp. 241–254.

[HKP92] Hertz (John), Krogh (Anders) et Palmer (Richard). – Introduction to the theory of neural computation. – Addison Wesley, Mars 1992.

[Kai91] Kaindl (H.). – Tree searching algorithms. *In : Computer, Chess, and Cognition*, éd. par Marsland (T.A.) et Schaeffer (J.), pp. 133–158.

[Kie83] Kierulf (A.). – Brand - an othello program. *In : Computer Game-Playing*, éd. par BRAMER (M.A.), pp. 197–208. – Hellis Horwood, 1983.

[Kie89] Kierulf (Anders). – New concepts in computer othello corner value, edge avoidance, access, and parity. *In : Heuristic Programming in Artificial Intelligence, the first computer Olympiad*, éd. par Levy (David) et Beal (D.F.), pp. 225–240.

[Leo95] Leouski (Anton). – Learning of position evaluation in the game of othello, 1995. ftp://ftp.cs.umass.edu/pub/techrept/ABSTRACT/UM-CS-1995-023.ABS.

[LM88] Lee (Kai-Fu) et Mahajan (Sanjoy). – A pattern classification approach to evaluation function learning, 1988.

[SA77] Slate (D.J.) et Atkin (L.R.). – Chess 4.5 - the northwestern university chess program. *In : Chess Skill in Man and Machine*, éd. par Frey (P.), pp. 82–118. – Springer-Verlag, 1977.

[Sch89a] Schaeffer (Jonathan). – Distributed game tree searching. *Journal of parallel and distributed computing*, vol. 2, n° 6, 1989, pp. 90–114.

[Sch89b] Schaeffer (Jonathan). – The history heuristics and alpha-beta search enhancements in practice. *IEEE transactions on pattern analysis and machine intelligence*, vol. 11, n° 11, 1989, pp. 1203–1212.

[Sch91] Schaeffer (Jonathan). – Checkers : a preview of what will happen in chess ? *Journal of the International Computer Chess Association*, vol. 2, n° 14, 1991, pp. 71–78.

[Sco69] Scott (J.J.). – A chess-playing program. *In : Machine Intelligence 4*, éd. par et D.Michie (B.Meltzer), pp. 255–265. – Edinburgh University Press, 1969.

[Wei89] Weill (J.-C). – *Contribution à la programmation des jeux de réflexion.* – Thèse de Doctorat, Université Paris 8 - Vincennes, octobre 1989.

[Wei91] Weill (Jean-Christophe). – Experiments with the negac* search, an alternative for othello endgame search. *In : Heuristic Programming in Artificial Intelligence, the second computer olympiad*, éd. par Levy (David) et Beal (D.F.), pp. 174–188.